Vol. 16, No. 9 • September 2000

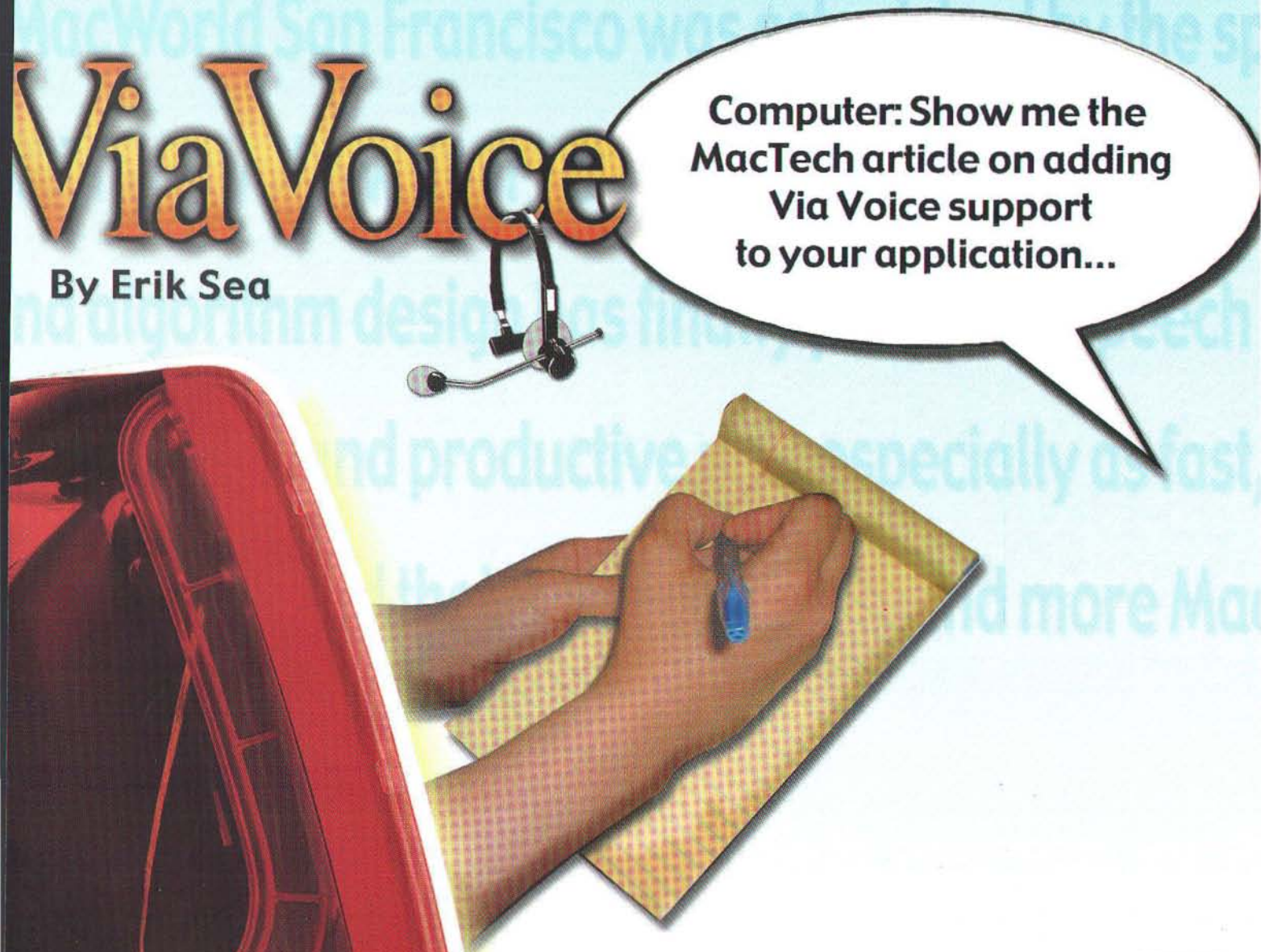
MacTech

The Journal of Macintosh Technology and Development

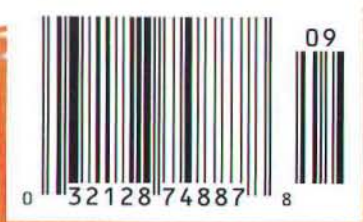
Inside: The Atomic Café

ViaVoice

By Erik Sea



$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



ORDER NOW!
Get the industry's #1 tool suite for developing Classic Mac® OS and Mac® OS X applications.
Call 800-377-5416.

Keeping Mac dreams alive since 1993.

Two great operating systems, one CodeWarrior.

Mac developers depended on CodeWarrior to make the platform architecture shift from 68K to PowerPC processors. Now CodeWarrior does it again, speeding your transition to the next new operating system. CodeWarrior for Mac OS, Version 6.0 supports development for both OS X and Classic Mac operating systems from a single, powerful, award-winning Integrated Development Environment. Discover how CodeWarrior for Mac OS, Version 6.0 can help you realize your Mac development dreams.
Visit **www.metrowerks.com/go/mac**.

CodeWarrior®

metrowerks®
Software Starts Here

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

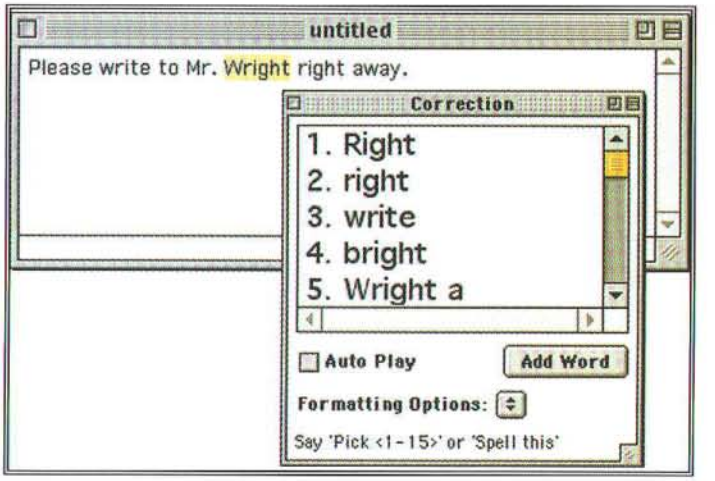
DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

September 2000 • Volume 16, Issue 09

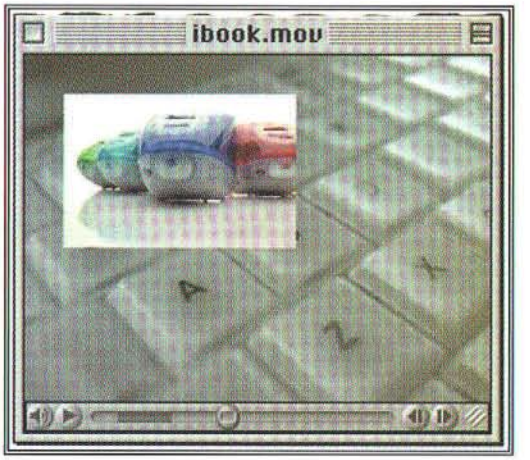
Feature Articles



Via Voice.....Page 42

Columns

OS X Page 52



The Atomic Café Page 22

By John C. "Hsoi" Daub, Contributing Editor. Austin, Texas USA

What We Can Learn From OpenBSD

Like the whole of the Mac community, I am eagerly awaiting the arrival of Mac OS X. Not only will we have the best user experience of any operating system available today, but we'll finally have the muscle under the hood to go places the Mac has never been before. Coupled with hardware like the dual processor Power Mac G4 and the Power Mac G4 Cube, we're now ready to tackle the big server and business markets, right? Well, almost.

During a particular daily pilgrimage to the Slashdot website, I happened upon a few articles about OpenBSD. From the OpenBSD.org web site: "The OpenBSD project produces a free, multi-platform, 4.4BSD-based Unix-like operating system. Our efforts emphasize portability, standardization, correctness, proactive security, and integrated cryptography." The security aspect of OpenBSD is what sets it apart from other operating systems; the OpenBSD project aspires to be number one in the industry for security, if they're not already.

Secure by Default

Mac users have long boasted about the Mac OS's "security by default". When the U.S. Army's websites were cracked June 28, 1999, the Army responded by switching to Macs. Events like these allow Mac users to put a feather in their cap. The Mac OS isn't uncrackable, but lacking a command line and not being Windows nor Unix-like, many of the potential vulnerabilities of an operating system simply don't exist. But wait a minute! Doesn't Mac OS X have a command line? And what about the BSD layer and other Unix-isms present in Mac OS X? Hrm. Perhaps it's time for the Mac community to pay more attention to security issues. A good place to start, especially for us developers, is to take a cue from the OpenBSD project.

One aspect of OpenBSD's security stances is to be "secure by default". That means the operating system is shipped with all non-essential services disabled. As a user becomes more familiar with the system and desires to utilize more services, he or she will have to learn about the process and what needs to be enabled. Hopefully by going through this process, the user is more likely to learn about security issues. By educating a user in a safe and forgiving environment, not only does it lead to a smarter user, but hopefully helps him or her avoid learning about security the hard way.

Granted OpenBSD's target audience is different than Mac OS's, so it's likely what services the two operating systems would provide by default would be different as well. But by the same token, the target audience for the Mac OS is more likely to be less computer savvy than your typical OpenBSD user. With broadband Internet access growing exponentially and more and more people getting online (recall those iMac sales numbers), it becomes even more critical to the Mac user experience to provide a safe and secure environment right out of the box. Remember, according to that iMac commercial there are only three (well, two) easy steps to get on the Internet: plug in, get connected; there's no step three. Being a security expert is not one of the steps.

Improve Code Quality

How many times in the past few years have you heard about security problems due to "buffer overflow?" Ultimately it's just a "simple coding error," but how many of these errors could have been caught and fixed if greater emphasis was placed on quality of code instead of hacking in twenty new features and shipping before the end of the quarter? The potential cost of that simple error could be far greater than the costs involved in having a solid code review and auditing process in place.

The proactive code auditing process utilized by the OpenBSD project isn't as much about looking for security holes as it is looking for coding bugs. They simply perform an extensive analysis of every source file. If new problems are found, then previously audited code gets reviewed again with the new problems in mind. Auditing the code multiple times by multiple people helps to improve not only the security of the code, but also the overall quality of the code. It's a nice double-benefit.

I understand the realities of software development: budgets, marketing requirements, schedules running over, being severely understaffed. Unfortunately due to these realities, quality of code is often sacrificed, which results in less than optimal product quality. And if you ship a shoddy product too many times, people will stop buying your products and lose faith in your company. The OpenBSD project's focus on quality allows them to proclaim at the top of their website that it's been three years without a remote hole and two years without a local hole in the default install. That's the sort of quality consumers are starting to expect these days. Instead of making a fuss over how Mac OS X won't crash if one application crashes, why don't we just have applications that don't crash in the first place? We won't be able to hide behind our disclaimers and licensing agreements forever.

So What Can We Learn?

The Mac OS X public beta should be released by the time you read this. If Apple has already taken steps towards being secure by default, all the better! If not, it is a beta, so that means there's time to fix it. But this isn't just a call for Apple to do something; this is a call to *you* to rethink your assumptions and consider the implications that come with our new OS paradigm. Every line of code needs to be written and reviewed with security and quality in mind.

If we want Apple, and hence our own businesses, to grow and flourish in the server and business markets, we need to think different from all the other players in that field. Except perhaps the OpenBSD project; their stance on security and quality is where we need to start thinking the same. MT

John C. Daub spends his days working as a developer for Aladdin Systems, Inc., currently working on the StuffIt Deluxe team. John spends his nights as he always does: playing with his wife and kids. You can contact John at hsoi@hsoi.com.

Thanx to James Chamberlain, Carl Constantine, Ron Davis, and Jim & Mary Ellen Lee for their input; and to Jessica for being such a sweetie. :-)

The key to thinking different...

Works on the iMac!

The New MacHASP USB Key!

Question: Is MacHASP USB* a software security key or a sales tool?

Answer: It's both!

MacHASP USB is the world's first software protection key for the iMac. It gives you sophisticated license enforcement and comprehensive protection against illegal use ... in other words, real security.

Then it gives you a big selling advantage.

With MacHASP USB, you can license multiple software modules and applications. You can instantly unlock or upgrade them in the field. Plus you can freely distribute demos.

Bottom line: MacHASP USB locks out illegal users and unlocks your full sales potential – without getting in anyone's way. Call now to request a Developer's Kit and our newly published guide to USB features and benefits.

*For all USB-equipped Macs running an OS with USB support. Fully compatible with the ADB version of MacHASP.

Mac OS

Accelerated for Power Macintosh

USA: 1-800-223-4277
212-564-5678
Int'l: 972-3-6362222

www.aks.com/mt

ALADDIN®
KNOWLEDGE SYSTEMS LTD

Mac software protection provider, Micro Macro Technologies, becomes part of Aladdin, giving its customers even better service from the number one name.

By John C. Welch

MacWorld New York 2000

The Network Manager's eye view

Welcome To The Show

The New York MacWorld Expo was a ground breaking show in many ways. Firstly, we have the attendance figure of 61,250 attendees. While well below this year's San Francisco number of over 85,000 attendees, it was a third more people than last year's New York MacWorld Expo, and well over the best figures for the best of the fabled Boston MacWorld Expos. A jump in the numbers like this is good for many reasons, but one of the important ones is that it's a sign that the New York move is finally picking up the numbers it needs to stand on its own, without being compared to shows from the Boston heyday. Attendance numbers like this year's also means that vendors who may have previously ignored New York for the larger San Francisco show will think twice about that next year. Finally, for the first time, there were no huge empty areas of show floor, not-so-cleverly hidden behind partitions.

Keynote

But enough cheerleading, what about the show itself? Well, from my point of view, it was excellent all the way around. The keynote was, as usual, an example of showmanship at its finest. Steve Jobs' famed Reality Distortion Field was working its usual magic, although considering the state of Apple the last few years, it no longer needs to work quite as hard. The hardware announcements were welcome indeed. As an administrator, I am very pleased to no longer have to automatically toss out the standard Apple mouse and keyboard that comes with the Macs we buy. And, although this may sound unusual for a corporate environment, the fact that the new mouse is very pleasing to the eye is a bigger bonus than you might think.

The announcement of the multi-processor G4 Macs was not that unusual to anyone who saw the demo of them at the Apple World Wide Developer Conference this past May. This was essentially a repeat of the initial demo of the G4 at the WWDC in 1999, and then the release not long afterwards. Although many will point out that the current MacOS can't handle multiple processors, that is not completely accurate. Apple has done a lot of work with the capabilities of the MP libraries in the current MacOS, especially with low-level tasks. Any developer taking advantage of these libraries should notice a definite improvement in performance on the MP Macs. This is also important for the upcoming public beta of MacOSX. By having MP Macs out in the hands of users prior to the beta, the radically improved MP capabilities of OSX will become immediately evident to Mac users, and this will help drive sales of not only MP Macs, but of OSX when it is released. I was surprised to see that Apple was not only making MP a standard feature of the mid-range and high-end G4 towers, but that they were able to do it for the same price as the Uniprocessor Macs. In essence, Apple is giving us a 'buy one, get one free' deal on the second CPU, and that is not a bad deal.

I'm also excited about the MP G4 models for more immediate reasons. The company I do most of my work for is a scientific firm. We make heavy use of Unix, multiple processors, and applications like Research System's IDL, which is a scientific image processing application. This type of application makes good use of additional processors when run under an operating system that supports SMP properly. Considering how expensive large-scale compute servers can be, the ability to have that level of processing capability on a scientist's desk for around $3000 has not only me, but many of my users eagerly awaiting the

John Welch <jwelch@aer.com> is the Mac and PC Administrator for AER Inc., a weather and atmospheric science company in Cambridge, Mass. He has over fifteen years of experience at making computers work. His specialties are figuring out ways to make the Mac do what nobody thinks it can, and showing that the Mac is the superior administrative platform.

public beta of OSX, and the next round of Mac purchases. Since we are a scientific firm we also have network data needs that would be considered insane by anyone who isn't a Photoshop pro. When you are dealing with atmospheric models, the data sets that support these models can easily approach, and often exceed the gigabyte size range. So finding out that Apple is making Gigabit Ethernet standard equipment for the new G4 towers turned my smile up yet another notch.

The new iMacs were, as always a crowd pleaser. The new colors are rich and vibrant, but somehow more dignified than the previous flavors. The new pricing structure is appealing to a corporate environment as well. To be able to get an iMac with firewire, VGA mirroring, (a very welcome new iMac feature), 192MB of RAM, the three year extended warranty, a USB Zip and SuperDrive for under $1700 makes buying one almost a no-brainer. Even when you add in things like Office98, and the standard utilities and packages we use, we are still talking about a silent full-featured corporate computer for under $2500. Considering that corporate price points are often in the $2500 to $3000 range, the new iMacs nail this particular sweet spot dead on. It's remarkably coincidental how a company that is constantly saying they have no interest in the corporate enterprise is making some of the best corporate computers around. Or maybe home users do have a desperate need for Wake-On-LAN that none of us I.S.-types have figured out yet.

I said the iMacs were almost a no-brainer for the corporate crowd, only because there is one other Mac that is even better suited for this arena: The Cube. True, it's not expandable, and you can't put in a different video card, and it has only space for one hard drive, and all peripherals have to be external. I still think this object d'art is going to be a big seller in the corporate area. First of all, it's gorgeous, and I mean in a Porsche/Ferrari kind of way. This thing just seems to ooze coolness. It looks like it should be on a shelf with other ridiculously expensive knick-knacks in a house on a prime-time soap opera. I can't think of a better computer, especially with one of the new flat-panel LCDs to put on a desk that is in an open area, or a receptionist's desk. The Cube is just, well, phat. Secondly, it's silent. Not quiet, but almost completely without noise. The hard drive is the only inherent noise coming from the Cube, and that is low enough to be ignored. For those of you who don't understand why this is important, think about the difference in your office or cubicle the next time you have to turn your computer off. I have a Beige G3 server, an external RAID box, and an 18-tape Digital Linear Tape autochanger in my office. Believe me, the thought of getting rid of any one of those noisemakers makes me smile. Now think about a corporate cube farm full of nice big Pentium towers, each with the standard two or three fans. Now imagine all that noise gone, to where you can hear the mouse clicks from a cubicle almost at the other end of the room. That is a cube farm full of Cubes; beautiful, isn't it?

While I will acknowledge that the Cube is totally inappropriate for someone who needs multiple processors, or specialized PCI cards, or for many server needs, the truth is, the vast majority of corporate users never add cards, never upgrade video cards, never do anything that the Cube cannot handle with ease. Considering how often people end up putting minitowers on their desktops, just because having to reach under their desk to get to the CD drive, or the Zip drive is such an annoyance, the size of the Cube is also going to be a big draw for the corporate crowd. Again, for a company totally disinterested in the corporate market, the Cube is one heck of a corporate computers. Apple's protestations here remind me of the "Brutus is an honorable man" speech from Julius Caesar.

That is not to say I found no shortcomings from the new hardware. I was dismayed to see Apple jumping to a brand new monitor connection with no way around the new system for those looking to combine older monitors with newer Macs and vice-versa. This is a temporary annoyance at best though, because I can already hear the peripheral companies firing up the VGA to ADC converter production lines. Still, it would have been nice to have an announcement to that effect.

The final bit of interesting keynote news was of course, OSX. Not the Aqua demo, (and that's what these are, Aqua demos. Outside of the developer community, I highly doubt that 90% of the Mac user community has seen any part of OSX other than Aqua), but rather the public beta news. The September time frame is not a hideous delay, and if it means a better beta experience for all, then I see it as a good thing. OSX still looks to be on track for the January/Early 2001 commercial release, and as long as that doesn't slip by more than a few months, OSX should be still considered to be close to on time. Were I to be a cynic, I would say that by having a few million users who are living with, and happy with OSX, the initial sales of the new OS should be a hit right away, as those folks are not going to want to go back to 9.X when their beta copies expire. But that would be cynical.

The Rest of the Show

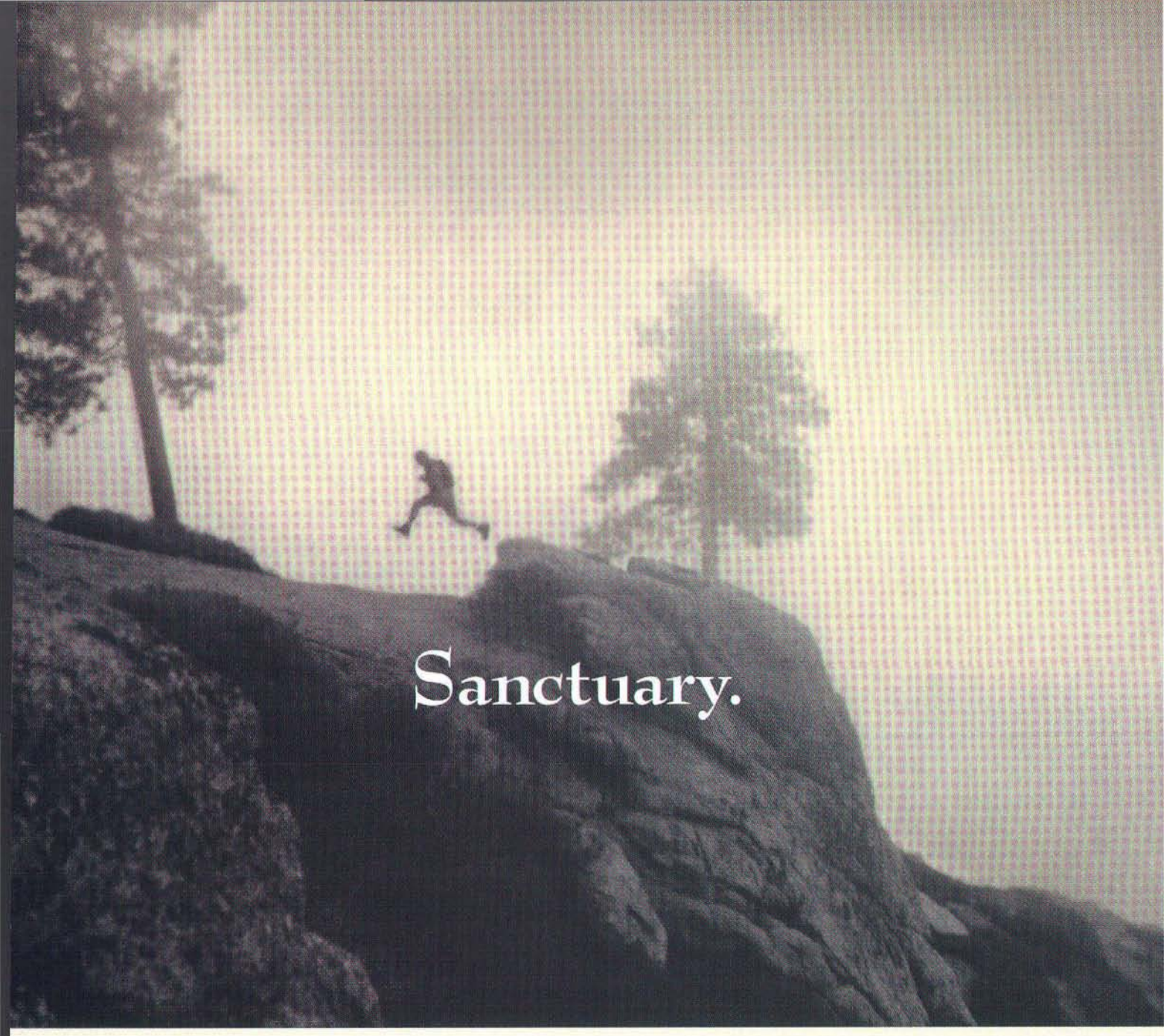
But the keynote is only the more visible part of MacWorld Expo, and only the beginning. There is a whole show floor, and three days of sessions that are of use to almost anyone.

Products

The first product of MacWorld that jumped out at me hasn't been released yet. That was the announcement by Tenon Intersytems of a full featured X Window application for OSX. This is a major announcement for the new OS, one that will help it gain real acceptance in the Unix community at large. X Window is the way that Unix computers can share applications that have a graphical interface with multiple users. This is not the same as products like VNC or Timbuktu. Those products are remote control applications, allowing you to take almost physical control of a remote computer, and use it almost as if you were sitting at that machine's physical keyboard. What the X Window System, (its proper name) allows is for multiple users to log into a single computer, and use a single application at the same time. The code in the application executes on the remote computer, and only screen, mouse, and keyboard data is sent between machines. There are X Window applications available for almost every computer system on the market, but it is especially important in the Unix world, as X is the primary means of running applications with a graphical interface. X is essentially a client-server display model, although its use of those terms can be a bit confusing. Essentially, the display functions are part of the X server, and the applications running, and sending display information to the X server are the X clients. These can be not just remote applications, but applications residing on the local hard disk of your computer.

Until this announcement, the only way to run X on OSX was to use a Darwin port of the XFree86 X Window application, (ported by John Carmack of Id Software, among others). However, the lack of a commercial X Window system for OSX was a serious hindrance to OSX's acceptance as a 'real' Unix-based OS. The Tenon announcement changed all that. What is just as important is that Tenon's product will allow OSX to serve standard X Window applications to other remote machines, and to the local user as well. This means that almost any BSD Unix application that is able to run on the version of BSD that is part of OSX, and can run on the G3 or G4 processor can run under OSX. This opens up a potentially huge library of applications that Mac users have had to run on other systems before. Especially for the Scientific and Technology market, or SciTech, full X capabilities for OSX is a serious point in the OS's favor at companies and universities in that arena. Finally, the Tenon announcement also covered that they will include X programming libraries and utilities with the product. This means that OSX-based developers will be able to create applications that take advantage of the G3 and G4 architectural advantages, but are available to anyone who can run X Window applications on their computer. So a developer could easily create an OSX application that had the Aqua interface, and took advantage of OSX's capabilities, and then with a little work, add in the capability for that same application to be run by almost any other computer platform in existence today. This is a huge step forward in compatibility for the Mac, and will have a very positive effect on the platform's growth.

Other products that caught my eye, although not to the same extent as Tenon's included the SANcube, from MicroNet Technology. The SANcube is interesting as an initial implementation of a FireWIre-based Storage Area Network, (SAN) device. The advantage to a SAN is that all storage on a SAN are independent of any host. That is, to get to disks on



Move to digital.forest. The leading Internet Service Provider for the Macintosh community.

You need maximum flexibility, security and performance from your ISP. You'll find it with digital.forest. We're the proven provider of Macintosh Internet service.

Founded in 1994, our specialty is Mac systems and technologies. In fact, digital.forest is the world's largest Macintosh Hosting Provider. And our hard-earned knowledge of Mac server software is the broadest in the business.

We pioneered Macintosh Server Colocation and FileMaker Pro database hosting services. And our commitment to service keeps growing. So whether you choose us to host your files or house your servers, you've chosen the best.

Go with the industry leader. Discover the sanctuary of digital.forest.

digital.forest

877-720-0483

info@forest.net

www.forest.net

Call toll-free in the U.S. and Canada.
For international inquiries, please call 425-483-0483.

FileMaker Solutions Alliance Associate

FileMaker Pro is a registered trademark of FileMaker, Inc. The digital.forest logo is trademark or registered trademark of digital.forest, Inc.

a SAN, you just mount the drives in the SAN device. No need to have a file server with the disks attached to it. This allows for greater flexibility in that you can have multiple computers with different OS's accessing the SAN storage device. It can also improve reliability by decoupling network storage from network servers, which means the storage on a SAN only has to be a storage device, not a file server with an OS, and the problems that a server can cause. Unfortunately, due to limitations in the current implementations of FireWire disk access methods, the disk access on the SANcube is controlled at the volume level. This means that if you have only a single partition on a SAN cube, and 5 servers that are attached to that SANcube, only one server can have write access to that partition, all others can only read from it. MicroNet recommends that you try to create a partition for each computer attached to the SANcube, thereby giving each computer on the SAN a partition that they have full access to, and read access to all the others. Hopefully, as FireWire is updated, this process will be able to happen at the file level, rather than the volume level. In any case, the SANcube still has great potential and usefulness, for things such as server farms, or situations where you only want one server to be able to write to a disk.

Two of my favorite Mac networking and administration companies were at the Expo as well, namely Thursby Systems and Alsoft. Thursby is the maker of such cross-platform compatibility products, such as DAVE, (for connecting Macs to a Windows network), MacNFS, (which gives Macs access to Unix files), and TSSNet, (to connect Windows machines to Mac Networks.) For those of us running multiple platform networks, Thursby's products are the way we avoid the host of connectivity problems that multiple platforms sometimes create. Alsoft is the company that makes DiskWarrior, which is most likely the best drive recovery tool on the Mac market. All of these products are an essential part of any networking toolkit, and I have used them in various combinations over the years, and consider them to be an absolute requirement for my administrator's toolkit. (On a side note, Chuck Goolsbee, keeper of the Mac-Manager's list, gave Alsoft one of the hard to come by Mac-Manager's list buttons, as a thanks for all the administrator's keisters that DiskWarrior has saved since its release. I was with Chuck when it happened, and considering how often DiskWarrior has saved 'dead' drives belonging to various executive types for me, I can't think of anyone more deserving.)

Sessions

As interesting and fun as the show floor is, there is yet another part of MacWorld Expo, that while not as glitzy, is a major part of the show for me, namely the various Pro, User, and preshow sessions.

The first set of sessions are the preshow conference workshops. These are seven hour, in-depth classes that cover everything from an introduction to Mac networking to Final Cut Pro. Not for the casual observer, the nice thing about the workshops is that with a seven hour timeframe, the presenters can really get into the kind of detail that a normal show session wouldn't allow them to. For me, the schedule is a little frustrating, so I tend to bounce between two or three different sessions. This year, I was having to decide between workshops on the Apple Macintosh Manager, a practical introduction to Mac Networking, and a getting started with AppleScript workshop. Although some might question the value of some of these workshops, especially the ones aimed at beginners, I would caution against that line of thought. These workshops, like almost all the rest of the sessions aren't taught by professional trainers, they are taught by the people in those fields, doing real work, and giving their experience back to you. As an example, the "Getting Started with AppleScript" session is run by Sal Soghoian, the AppleScript Product Manager for Apple. Even though I would consider myself to be a fairly competent AppleScript programmer, Sal always manages to give me something new, either in techniques, or looking at AppleScript in a different way. I have found this to be true of almost every workshop I have ever attended, so again, I recommend thinking before dismissing a 'beginner' workshop.

The next set of sessions are the Pro Conferences, and these are more targeted to a specific part of the Expo audience. The two that appeal to me as a network administrator are the Mac Manager Track, and the Macintosh Networking and Communications Track. The Mac Manager Track deals with, well, managerial issues. Things such as license compliance, backup strategies, an update on Apple's Open Source initiatives. I was involved with two of the sessions in this track, one titled, "Becoming a Successful Mac Manager Part I — Business Issues", and the other was titled "Scripting Mac Admin Tasks". The first dealt with techniques for communicating within a company that is considering moving to a single platform, which is not the Mac, and the second dealt with how to integrate AppleScript into your network administration tasks. (I'll withhold an opinion on these two, as I am somewhat biased.). Another session which was less technical than some others, but still very useful was the part II session to becoming a successful Mac manager. This part dealt with the nitty gritty of being a Mac I.S. manager. Things like operational procedures, hiring, internal promotion, turnover reduction, etc. All things that may not be as technically exciting as the new Macs, but are quite necessary to running a network or an IS department. Just to point out the experience of some of the presenters, one of the folks presenting this particular session was Chuck Goolsbee, who besides his role with the Mac-Manager's list, is also VP of Technical Operations for digital.forest, Inc, one of the biggest Mac web-hosting companies in the U.S. Again, where else can Mac managers get the benefits of that kind of experience, live, other than MacWorld?

In the Networking and Communications track, there were some very interesting sessions as well. The first one that jumps to mind is the Mac Networking Update session. One of the presenters for this one was Thomas Weyer, the Networking and Communications Manager for Apple. This was invaluable to me, not only for an idea of the direction Apple is taking in this area, but for practical tidbits as well. One of the most useful was Tom explaining how to set up AirPort Base Stations to get proper channel separation so that you have cleaner signal on your wireless network. Or the explanation that the dot graph for the AirPort Control Strip module is a measurement of the signal-to-noise ratio that you are currently getting for your AirPort device, not connection speed or signal strength as has been assumed. For those of us with wireless networks, these are small, yet invaluable bits of information. Other sessions included going from an AppleTalk to a TCP/IP based network, tips on tuning Open Transport, information on integrating MacOSX into existing networks, a Network Managers Forum, internet security advice, Virtual Private Network information, and how to use Kerberos authentication with the Mac OS.

But again, it isn't just the session titles or subjects that make these invaluable, it's the fact that they are presented by people who either create the technology we use, or live with it. I was able to get more VPN information in 10 minutes of discussion with Bill Vlahos, who, in addition to being the presenter for the VPN session, is also a Network Services Engineer for NASA's Jet Propulsion Laboratory. He deals with, and lives with cross-platform VPN issues every day, so his answers were based on experience and real world knowledge, instead of marketing brochures. The session on Comparing AppleShareIP with Windows NT and Windows 2000 services was given by Paul Nelson, VP of Engineering for Thursby Software Systems, who are the makers of DAVE, a Windows networking stack for the Mac. So here you have a session given by someone who understands not only Mac networking, but Windows networking at an extremely low level. Again, the experience brought to the table here is not going to be replicated by a professional trainer. My only complaint with the sessions is that I haven't figured out how to clone myself, so I can't possibly make it to all of them.

Conclusion

In the end, even though MacWorld expo is billed as a consumer exposition, there is enough good information and products there for any network manager or administrator that deals with Macs to justify the cost. As well, if you have some knowledge that you would like to share, the Expo is always looking for new presenters and new ideas. I have yet to be disappointed by a MacWorld, and this one was no exception. From the keynote to the show floor, to the sessions, there was enough there to keep my I.S. heart as happy as could be. The next MacWorld, in this country at least is in San Francisco, in January, 2001. I hope to see some of you there. MT

By Tom Djajadiningrat

3D For Free Using the Mac's Standard Apps

Converting raw 3D text files to QuickDraw 3D's 3DMF format using AppleScript

Summary

This article introduces you to the basics of 3D files in general and the 3D Metafile (3DMF), QuickDraw 3D's native 3D format, in particular. It shows how you can use AppleScript to easily convert a raw 3D text file into a 3DMF readable by the QuickDraw 3D Viewer or any other QuickDraw 3D compatible application. With this knowledge you can make hand-written data or data exported by a spreadsheet or database application suitable for visualisation as standalone 3D models.

Introduction

The combined power of the suite of goodies that comes standard with your Mac is really quite amazing. One such goody is the QuickDraw 3D Viewer. The QuickDraw 3D Viewer makes it possible to do basic viewing of 3D models that are formatted as 3D Metafiles (3DMF), QuickDraw 3D's native file format. Since the QuickDraw 3D Viewer is supported by SimpleText, which comes standard with a Mac, files written in 3DMF format can be viewed on any PowerMacintosh running QuickDraw 3D. Maybe you would like to visualise your spreadsheet or database data as standalone 3D models or perhaps you would like to write your own low-polygon count models by hand rather than use a 3D modeller. To achieve this you could make the spreadsheet or database application export complete 3DMFs. Often though it is quite awkward to get the formatting right. And though it is possible to write complete 3DMFs by hand, you of course want to reduce repetitive chores, such as adding brackets, to save time and reduce errors. Here we take the approach of writing minimalistic 3D files which are then post-processed into 3DMF. So how do we do this post-processing? Enter two other Macintosh standard goodies: AppleScript and the ScriptMaker application. Although it is seldom used for this express purpose, AppleScript is actually quite good at reading and writing text files. Another nice feature of AppleScript is the ease with which we can make a droplet: a script which performs certain actions on documents which are dropped on to it. This allows us to convert multiple files with the ease of drag and drop.

QuickDraw 3D's 3DMF Format

3D models can be described using vertices and faces. A vertex is a point in 3D space described by x, y and z co-ordinates. A face is a polygon of which each corner point is a vertex. To prevent unpredictable results a face should be planar: all its vertices should lie in one plane. **Figure 1** shows a technical sketch of a table which consists of a square face (the table top) and eight triangular faces (the legs). **Figure 2** shows the corresponding 3D model in the QuickDraw 3D Viewer.

While walking about at the Geneva Motor Show, **Tom** spotted a Japanese concept car in which a display clearly showed a Macintosh alert box with an error of Type 2. As he does not find the prospect of having to install and configure Linux for car navigation, car audio and engine management particularly appealing, he can't wait for the release of OSX.

// Like most Mac developers,

// I easily spend 12 hours a day

// staring at line after line of C++ code in tiny, 9 point Monaco.
// Sometimes it makes my eyes feel like they're on fire.

{
So the last thing I need is some
fuzzy monitor that adds to my headaches.
}

/****** begin excitement ******/

// That's why I'm so jazzed about this SGI monitor.

// Its ultra-high resolution = 1600 x 1024 and dpi = 110,
// giving me razor-sharp contrast.
// And the high refresh rate is
// perfect for poring through lines of code.

// At first, I was amazed at the clarity, the fine details that emerged.

{
It was like seeing things for the first time.
}

// Later, though, I learned to appreciate the wide aspect ratio (= 16:10),
// with a generous 17.3 inches of viewing area. SGI's 1600SW
// lets me have all my documents viewable at once, and it's
// a flat panel so it fits on my desk with room to spare.

// From the moment I saw this thing I was hooked. You will be, too.

{
Especially when you find out
how affordable it is.
}

// Check it out.

/****** end excitement ******/

Michael Whittingham
V.P., Engineering
Wired, Inc.

www.sgi.com/flatpanel

This advertisement was written by an actual engineer. This is the first time in 7 months that he's been seen during the day.

sgi™

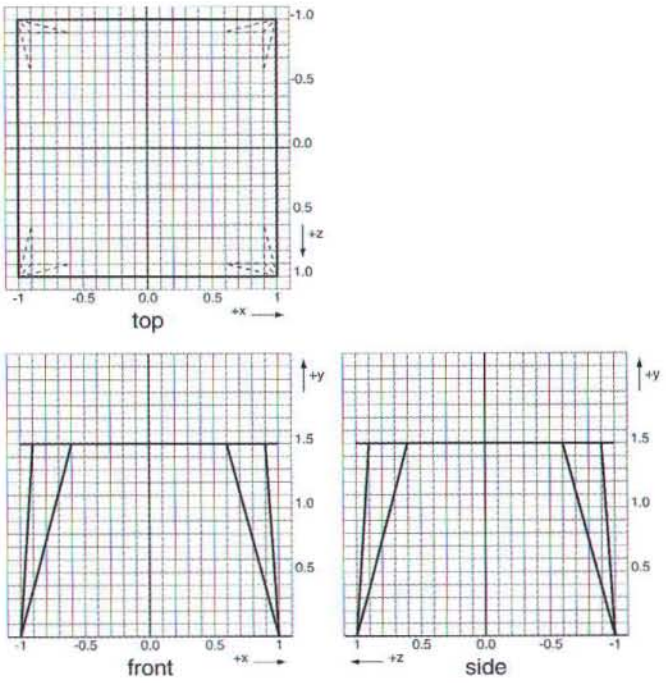


Figure 1. *Technical sketch of a table.*

Figure 2. *The 3D model in the QuickDraw 3D Viewer.*

Listing 1 shows a 3DMF file for this table. For our example we use the QuickDraw 3D mesh geometry, which is probably the easiest to understand way to describe a 3D model. The hash sign (#) marks a comment in a 3DMF. Everything on a line after a hash is ignored. The 3DMF here consists of two parts. The first part is the header, 3DMetafile (1 6 Normal toc>). The second part is formed by a container which contains a single mesh object. Let's look at the header first. The header, formed by the first line of the file, tells us that we have a 3D Metafile for QuickDraw 3D 1.6. The major version number is 1 and the minor version number is 6. The third field, Normal, is the type of 3D Metafile. The fourth field of the header, toc>, is a file pointer to a table of contents which is non-existent in this 3DMF. Don't worry about the type of 3D Metafile and the table of contents parameters. You do not need to know their exact meaning to understand this article and later we explain where to read up on the full 3DMF specification.

Listing 1: a 3DMF file

table.3df

```
3DMetafile ( 1 6 Normal toc> )
#This is a table with a square top and four legs
Container (
  Mesh(
    20 #number of vertices
    1 1.5 -1 #0
    -1 1.5 -1 #1
    -1 1.5 1 #2
    1 1.5 1 #3
    1 0 1 #4
    0.9 1.5 0.6 #5
    0.9 1.5 0.9 #6
    0.6 1.5 0.9 #7
    1 0 -1 #8
    0.6 1.5 -0.9 #9
    0.9 1.5 -0.9 #10
    0.9 1.5 -0.6 #11
    -1 0 -1 #12
    -0.9 1.5 -0.6 #13
    -0.9 1.5 -0.9 #14
    -0.6 1.5 -0.9 #15
    -1 0 1 #16
    -0.6 1.5 0.9 #17
    -0.9 1.5 0.9 #18
    -0.9 1.5 0.6 #19
    9 #number of faces
    0 #number of contours
    4 0 1 2 3 #0
    3 4 5 6 #1
    3 4 6 7 #2
    3 8 9 10 #3
    3 8 10 11 #4
    3 12 13 14 #5
    3 12 14 15 #6
    3 16 17 18 #7
    3 16 18 19 #8
  )
  Container (
    AttributeSet ( )
    DiffuseColor (1.0 1.0 0.0) #r g b
  )
)
```

Now let's look at what takes place within the bracket of the container. The container contains a mesh and another container with a diffuse colour attribute in it. First the mesh description lists the number of vertices involved. In our case there are twenty vertices. What follows is a list of vertices which QuickDraw 3D sees as numbered from 0 to 19. Each vertex is described by three co-ordinates. For example, vertex number 0 has the co-ordinates x=1, y=1.5 and z=-1. Next is the number of faces, eight in this case, followed by the number of contours. A contour is a polygonal hole in a face. Since there are no holes in our table the number of contours is 0. The mesh description finishes with the eight faces. The first number of a face description is the number of vertices involved in the face, the remaining numbers specify these vertices. For example, the square table top is face number 0, which has four corners formed by the vertices 0, 1, 2 and 3. Finally, there is a diffuse colour attribute which applies to the whole mesh. It is specified in red, green and blue components, each ranging from 0.0 to 1.0.

The Raw 3D File

It is very important to carefully consider what the raw 3D file should look like before you start writing a script to convert it to 3DMF. If you yourself export the file or write it by hand you are of course in complete control. If someone else does this for you—say an expert in low-polygon count models — you want to make sure this person formats the file in such a way that makes life as easy on you as possible. On the other hand, you may wish to respect this person's way of formatting files. Listing 2 shows an example of what a raw 3D file might look like. There are some differences between this raw 3D file and the 3DMF file we just discussed which make conversion pretty awkward, but it is understandable why the formatting of the raw 3D file is convenient for the person who writes the file by hand. Because of the vertex and face labels on each line it is immediately clear whether one is in the vertices or faces section when quickly scrolling through the file. Because of the commas it is possible to put spaces between a minus sign and a co-ordinate so that the minus signs line up vertically. This makes it easier to spot co-ordinates with the wrong sign. Both vertices and faces start rather than end with their index number so that the indices line up too. A face line does not include the number of vertices which saves work and reduces errors. Careful deliberation with all involved can safe much work and frustration.

Listing 2: A raw 3D file

table.txt

```
Description:This is a table with a square top and four legs

Vertices:20
Vertex: 0,  1, 1.5, -1
Vertex: 1, -1, 1.5, -1
Vertex: 2, -1, 1.5,  1
Vertex: 3,  1, 1.5,  1

Vertex: 4, 1,   0,   1
Vertex: 5, 0.9, 1.5, 0.6
Vertex: 6, 0.9, 1.5, 0.9
Vertex: 7, 0.6, 1.5, 0.9

Vertex: 8, 1,   0,   -1
Vertex: 9, 0.6, 1.5, -0.9
Vertex:10, 0.9, 1.5, -0.9
Vertex:11, 0.9, 1.5, -0.6

Vertex:12, -1,   0,   -1
Vertex:13, -0.9, 1.5, -0.6
Vertex:14, -0.9, 1.5, -0.9
Vertex:15, -0.6, 1.5, -0.9

Vertex:16, -1,   0,   1
Vertex:17, -0.6, 1.5, 0.9
Vertex:18, -0.9, 1.5, 0.9
Vertex:19, -0.9, 1.5, 0.6

Faces: 9
Face: 0, 0, 1, 2, 3
Face: 1, 4, 5, 6
Face: 2, 4, 6, 7
Face: 3, 8, 9, 10
Face: 4, 8, 10, 11
Face: 5, 12, 13, 14
Face: 6, 12, 14, 15
Face: 7, 16, 17, 18
Face: 8, 16, 18, 19

DiffuseColor:1.0 1.0 0.0
```

AppleScript

The next thing we do is to write an AppleScript to convert the raw 3D text file to 3DMF. In the folder <your harddisk>:Apple Extras:AppleScript you find the application ScriptMaker which you can use for writing AppleScripts. First we discuss the structure of our script, then we fill in the details.

The structure

Listing 3 shows the structure of our conversion script Convert to 3DMF. We use the on open construct so that we can use the script as a droplet. The file specifications of the files dropped onto the droplet appear in the list inDocList. We traverse the list inDocList through a repeat command and act on each file in turn. First we do the most basic of error checking. Within a try statement we check whether the file is of type 'TEXT'. If it is not, we bail out by throwing an error which leads to execution of the on error portion of the try statement. If it is a text file we call the parse handler. Within the parse handler we again have a try statement in which we try to extract the relevant information from the raw 3D text file and format it according to 3DMF standards. If all goes well we call the writeResultFile handler with as arguments the file specification of the raw 3D text file and the text of the 3DMF file to write. Within the writeResultFile handler we try to create a text file within the same directory as the raw 3D text file and the same name, except for the extension which we make 3df. Now that we have some idea of the structure of our script, let's turn our attention to the parse handler.

Listing 3: The structure of Convert to 3DMF

```
on open (inDocList)

  — traverse the list of files that were dropped on the droplet
  repeat with theFilePath in inDocList

    try
      tell application "Finder"

        —minimal error checking:
        —check whether the file is of type text.
        —if it isn't, throw an error
        if (get file type of file theFilePath)≠"TEXT" then
          error "This is not a text file."
        end if

      end tell

      —if we get here the file type is "TEXT"
      —and we parse the file.
      parse(theFilePath)

    on error inErrorText
      —handle any errors that may occur.
      display dialog ("An error has occurred:" ¬
                      & inErrorText)
    end try

  end repeat

end open

on parse(inFilePath)

  try
    —Here we try to read the raw 3D file,
    —extract the relevant information,
    —and format it into 3DMF format.
    <code omitted>

    —Call a handler to write the result to a file.
    writeResultFile(inFilePath, theOutputString)

    on error inErrorText
      display dialog ("An error has occurred:" & ¬
                      inErrorText)

  end try

end parse

on writeResultFile(inFilePath, inOutputString)

  try
    —Here we try to create a new text file
    —in the same directory as the raw 3D file,
    —only with the extension '3df' instead of 'txt'.
    —and write our formatted result to it.
    <code omitted>

  on error inErrorText
    display dialog ("An error has occurred:" & inErrorText)

end writeResultFile
```

The Parse handler

For the code of the parse handler please refer to Listing 4. First we read all of the raw 3D text file into a list called theLineList using returns as delimiters. This means that each element of the list theLineList contains one line of the raw 3D text file. We then traverse this list of lines using a repeat statement and look at the start of each line. There are number of possibilities.

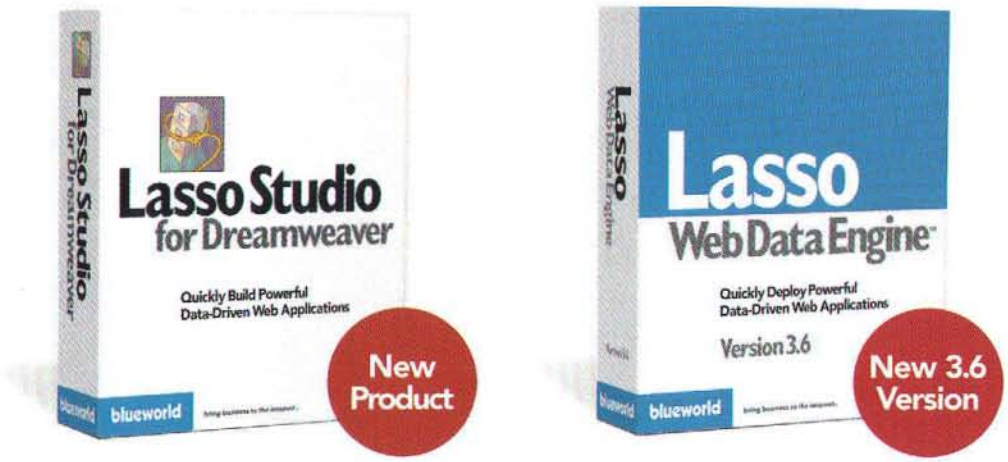
If a line starts with "description" we are dealing with the first line of the raw 3D text file which holds a description. Although this may seem superfluous as the file has a descriptive name, it can be a convenient location to store a long (>32 characters) description of the 3D object in the file. We ignore the word "description" and the colon and store the rest of the line in the string variable theDescription.

If a line starts with "vertices" we have come across the line which signals the start of the vertices description and which tells us the number of vertices in the file. We simply ignore this line. Instead of grabbing the number of vertices from this line we will count the vertices as we encounter them.

If a line starts with "faces" we are dealing with the line which signals the start of the faces description and which tells us the number of faces in the file. Again we ignore this line as we will also count the faces as we encounter them.

If a line starts with "vertex" we have a line with a vertex description. We ignore everything on this line up to and including the first comma which means that we eliminate the word "vertex" and the number of the vertex. The remainder of a vertex description line needs some filtering. We traverse this remainder looking at each character in turn. We replace the commas by spaces and eliminate superfluous spaces. For this last action we need to keep track of the last character using the variable theLastChar. For example, if there is a space between a

Quickly Build and Deploy Powerful Data-Driven Web Applications



Lasso Studio and Lasso Web Data Engine™ Lead The Way.

Building custom and shrink-wrapped database-driven Web applications requires a whole new way of doing things. Having pioneered the Web Data Engine™ over three years ago, Blue World and the Lasso Web Data Engine consistently lead the way providing a feature set Web developers describe as "incredible." Build online stores, discussion forums, resource management systems and other demanding database-driven Web applications with unrivaled performance, ease, security, extensibility, control and flexibility. Develop using multiple languages—including LDML, CDML, Server-Side JavaScript, Java and XML— and deploy across multiple platforms. Your Lasso code works identically regardless to which database you're connected. Lasso solutions for FileMaker® Pro databases easily scale to big iron ODBC-compliant databases like Oracle, Informix, Sybase and more with little or no change. What's more, the Lasso Java Application Programming Interface (LJAPI) provides developers an easy-to-use Java-based API for unprecedented extensibility.

Find out why hundreds of thousands of websites rely on award-winning Lasso technology for their business critical Web data. Download a 30-day evaluation copy at www.blueworld.com/download/ or order securely online today at the Blue World Store at store.blueworld.com.

Lasso Product Line – The leading Web tools for Macintosh and beyond.

bring business to the internet™

blueworld

minus sign and a comma we eliminate that space. Also, if there are multiple consecutive spaces we reduce them to a single space. The characters we wish to keep are appended to the variable theVerticesString. After each line we add a comment with the index number of the vertex. By the time we have finished parsing all the vertices theVerticesString holds nice, clean versions of all the vertices in the file.

If a line starts with "face" we are dealing with a face description. The filtering of a face description is pretty similar to that of a vertex description. Again we ignore everything on this line up to and including the first comma. This means that we have got rid of the word "face" and the number of vertices involved in the face. Unlike with a vertex line, we need not worry about decimal places. What we do need to do is count the number of vertices involved in a face. If a digit is preceded by a space or a comma we increment the variable theNumberOfVerticesInFace. The characters we wish to keep are appended to the variable theFacesString. After each line we add a comment with the index number of the face. By the time we have finished parsing the faces theFacesString holds a nicely formatted version of all the faces in the file.

If a line starts with "DiffuseColor" we are dealing with the colour specification. We ignore the word "DiffuseColor" and the colon and store the rest of the line in the string variable theColor.

If a line does not conform to any of the previous options, then we either have an empty line or a line that we do not currently support. These lines are simply ignored.

In its current form, the script does not cater for holes. The number of contours is simply set to 0.

The rest of the parse handler assembles the various parts into a string variable theOutputStr. All we need to do now is write our freshly formatted string theOutputStr to a file through the handler writeResultFile.

Listing 4: Parsing and formatting

parse

```
on parse(inFilePath)

  —initialise the number of vertices
  set theNumOfVerts to 0

  —initialise the number of faces
  set theNumOfFaces to 0

  —the string with vertex descriptions
  set theVertsStr to ""

  —the string with face descriptions
  set theFacesStr to ""

  try
    —Copy the content of the text file into a list.
    —Use return as the delimiter.
    —This makes each list element one line.
    copy (read inFilePath as list using delimiter return)¬
        to theLineList

    — Traverse the list of lines.
    repeat with i from ¬
      1 to the number of items in theLineList

    set theLine to item i of theLineList

    if theLine contains "Description" then
      —Skip the label "description" and copy the rest.
      set theDescription to characters ¬
        ((the offset of ":" in theLine) + 1) ¬
          thru (the end of theLine) of theLine as string

    else if theLine contains "Vertices" then
      —Signals the start of the vertices list: do nothing

    else if theLine contains "Faces" then
      —Signals the start of the faces list: do nothing.

    else if theLine contains "Vertex" then
      —This is a vertex description

      —Track the last character we read.
      set theLastChar to ""

      —Traverse the line from the comma to the end of the line
      repeat with j from ((the offset of "," in ¬
          theLine) + 1) to the number of characters¬
          in theLine

        copy character j of theLine to theChar

        if "1234567890-." contains theChar then
          —These are valid characters which we keep
          set theVertsStr to theVertsStr & theChar
        else if theChar = "," then
          —We replace commas by spaces
          set theVertsStr to theVertsStr & " "
        else if theChar = " " then
          if theLastChar = "" then
            —Line starts with a space : don't copy the space
          else if theLastChar = "-" then
            —space preceded by a minus sign: don't copy the space
          else if theLastChar = " " then
            —Two consecutive spaces: don't copy the " "
          else if theLastChar = "," then
            —space preceded by a comma: don't copy the space
          else
            —This is a space we wish to keep.
            set theVertsStr to theVertsStr & " "
          end if

        end if

        — Keep track of the last character
        copy theChar to theLastChar
      end repeat

      —We append a comment with the vertex number to this line.
      set theVertsStr to theVertsStr & ¬
        " " & "#" & theNumOfVerts & return

      —Increment the number of vertices we have dealt with.
      set theNumOfVerts to theNumOfVerts + 1


    else if theLine contains "Face" then
      —This is a face description.

      set theCurrFaceStr to ""
      set theLastChar to ","
      set theNumOfVertsInFace to 0

      —Skip everything upto and including the first comma.
      —Traverse the line from the comma to the end of the line
      repeat with j from ((the offset of "," in ¬
                  theLine) + 1) to ¬
                  the number of characters in theLine

        copy character j of theLine to theChar

        if "1234567890" contains theChar then
```

```
            —These are valid characters which we keep
            copy theCurrFaceStr & theChar to ¬
                  theCurrFaceStr

            —If this valid character is preceded by a comma or a space.
            —then it must be a new vertex number.
            if theLastChar = "," or theLastChar = " " then
              copy theNumOfVertsInFace + 1 to ¬
                theNumOfVertsInFace
            end if

          else if theChar = "," then
            —We replace commas by spaces
            set theCurrFaceStr to theCurrFaceStr & " "

          else if theChar = " " then

            if theLastChar = "" then
              —Line starts with a space: don't copy the space
            else if theLastChar = "-" then
              —"" preceeded by a minus sign: don't copy the space
            else if theLastChar = " " then
              —Two consecutive space s: don't copy the ""
            else if theLastChar = "," then
              —space preceded by a comma: don't copy the space
            else
              —This is a space we wish to keep.
              set theCurrFaceStr to theCurrFaceStr & " "
            end if

          end if

          —Keep track of the last character
          set theLastChar to theChar

        end repeat

        —Build a line with a face description.
        set theCurrFaceStr to theNumOfVertsInFace & ¬
          " " & theCurrFaceStr & " " & "#" & theNumOfFaces

        —Build the description of all the faces.
        set theFacesStr to theFacesStr & ¬
          theCurrFaceStr & return

        —Increment the number of faces.
        set theNumOfFaces to theNumOfFaces + 1

      else if (item i of theLineList) ¬
        contains "DiffuseColor" then

        set theColor to characters ¬
          ((the offset of ":" in theLine) + 1) ¬
            thru (the end of theLine) of theLine as string

      else
        —Probably an empty line or something unsupported.

      end if

    end repeat

    —There are no holes in our model so the number of contours is 0.
    set theNumOfContours to 0

    —The 3DMF header.
    set theHeader to "3DMetafile(1 6 Normal toc>)" & return

    —The description
    set theDescription to "#" & theDescription & return

    —The attribute set
    set theAttributeSet to "AttributeSet ( )" & return & ¬
      "DiffuseColor (" & theColor & ")" & " #r g b" &¬
      return

    —We've got the parts, now assemble the 3DMF file
    —in the string variable theOutputStr.
    set theOutputStr to ¬
      theHeader & ¬
      theDescription & ¬
      "Container (" & return & "Mesh(" & return & ¬
      theNumOfVerts & " #num of vertices" & return & ¬
      theVertsStr & ¬
      theNumOfFaces & " #num of faces" & return & ¬
      theNumOfContours & " #num of contours" & return & ¬
      theFacesStr & ¬
      ")" & return & ¬
      "Container (" & return & ¬
      theAttributeSet & ¬
      ")" & return & ¬
      ")"

    —Call a handler to write the result to a file.
    writeResultFile(inFilePath, theOutputStr)

    —Done!
    beep

  on error inErrorText
    display dialog ("An error has occurred: " & ¬
                    inErrorText)
  end try

end parse
```

The writeResultFile Handler

Now that we have the string theOutputStr all that remains to do is to write it to a text file. This happens in the handler writeResultFile (Listing 5). In the inFilePath parameter we pass the handler the file path of the raw 3D text file. We want to create a finished 3DMF in the same directory and with the same name as the raw 3D text file but with a different file extension, which we want to be .3df. If the raw text file has

the file extension .txt we chop it off first. Using the newly created file path theOutfilePathStr we open a file with write permission. We then write our string inOutputStr to this file and set its type and creator. The type is 3DMF and here we use SimpleText as the creator though of course you can change the creator to the application of your choice. Finally, we close the file.

Listing 5: Writing our result to a file

writeResultFile

```
on writeResultFile(inFilePath, inOutputStr)

  —Cast the file path to a string.
  copy inFilePath as string to theFilePathStr

  —we write the resulting 3DMF file to the same directory as our original text file.
  — if the file path ends with .txt cut off the .txt
  if (theFilePathStr) ends with ".txt" then
    set theOutfilePathStr to (characters 1 thru ¬
      ((the offset of ".txt" in theFilePathStr) - 1) ¬
        of theFilePathStr) as string
  else
    set theOutfilePathStr to theFilePathStr
  end if

  —Append .3df to the file path in case of use by 'the evil empire'.
  set theOutfilePathStr to theOutfilePathStr & ".3df"

  —Create a file to write our result to.
  copy (open for access file theOutfilePathStr ¬
    with write permission) to theOutFileRef

  try

    —Write the output string to the file.
    write inOutputStr to theOutFileRef

    —Set type and creator.
    tell application "Finder"
      —Set the type to 3DMF, the file type of a 3D Metafile.
      set file type of file theOutfilePathStr to "3DMF"

      —We want the file to be openable by SimpleText (creator "ttxt").
      —But of course others are possible:
      —FormZ (creator "VTVS")
      —Geo3D (creator "3Tt*")
      —3DMFOptimizer (creator "OP20")
      set creator type of file theOutfilePathStr to "ttxt"
    end tell

    —Close the result file
    close access theOutFileRef

  on error inErrorText
    —Close the result file
    close access theOutFileRef
    display dialog ("An error has occurred: "¬
                    &inErrorText)

  end try

end writeResultFile
```

Finishing and trying out our script

We finish our script by turning it into a droplet. Choose save as from ScriptMaker's file menu, enter the name Convert to 3DMF and choose classic applet from the structure popup menu. Now drop the raw 3D text file table.txt on the Convert to 3DMF droplet. After a brief pause the Mac beeps and a file called table.3df appears in the same directory as table.txt. Double click the file. SimpleText should open and the QuickDraw 3D Viewer should display our table as shown in Figure 2. If you are unsure how to use the controls of the QuickDraw 3D Viewer, check out Balloon Help as it is quite helpful. On some systems the file may not open correctly after a double click in the Finder and shows as a QuickTime movie instead. This can be solved by launching SimpleText and choosing table.3df from the File Open dialog box. As this is of course awkward if you would like to open multiple files, Listing 6 shows a small script for a droplet which correctly opens 3DMF files that are dropped onto it . Note that if you use a non-English language system, AppleScript may ask you to locate your localized copy of SimpleText which is hiding on your harddisk under a different name.

Listing 6: Open3DMF

```
on open (inDocList)

  repeat with theFileSpec in inDocList

    tell application "SimpleText"
      open theFileSpec
      activate
    end tell

  end repeat

end open
```

Discussion

In this article we showed you how you can get your data into QuickDraw 3D's 3D Metafile format. We barely scratched the surface but now that you are up and walking you can teach yourself how to run using the following pointers.

Quesa: QuickDraw 3D to the future?

Quesa (www.quesa.org) is the astounding open source effort to build a 3D graphics library which offers binary and source level compatibility with QuickDraw 3D. If you are hesitant to support QuickDraw 3D because of Apple's announcement to drop the technology, you may wish to investigate Quesa. Quesa is not only suitable for Mac OS8/9 and MacOSX, it is completely cross-platform with support for Windows and Linux. A port for Be is expected.

Learning more about 3DMF

You can find a well organised collection of links to QuickDraw 3D documentation, including the 3D Metafile 1.5 Reference documentation, at www.quesa.org/other/links.html. A good read on a rainy sunday afternoon, the 3D Metafile documentation tells you in approximately 250 pages everything there is to know about 3DMF. You should pay special attention to these two aspects of the 3D Metafile:

- Alternative geometry types
  For our file we have used QuickDraw 3D's mesh geometry. The advantage of the mesh is that it is easy to

understand and write and that there are many QuickDraw 3D calls for mesh editing. Its big disadvantage is that it is rather inefficient when it comes to rendering. If you are mainly interested in fast rendering you may wish to look into two other geometry types: the polyhedron and the trimesh.

Another option is to use 3DMF Optimizer by Pangea Software to convert your meshes into trimeshes (www.pangeasoft.net/downloads.html).

- Binary vs. ASCII
  So far we have been writing 3DMF ASCII files. There is also a binary 3DMF format. The binary format results in smaller files which can be read faster by QuickDraw 3D applications. Luckily, there are conversion utilities around which convert our 3DMF ASCII file to a 3DMF binary file. One such utility is Anatas by Stefan Huber (www.topoi.ch). It converts from ASCII 3DMF to binary 3DMF and vice versa.

Better tools for AppleScript

While ScriptMaker is a great freebie and quite adequate for writing small AppleScripts, you will start to notice its limitations as your scripts start to grow. There is no built-in search and replace, variable watching or step-by-step debugging. If you feel you need a better AppleScript development environment have a look around at Developer Depot (www.devdepot.com). There you will find some professional AppleScript tools.

Better 3D tools

As the name implies the QuickDraw 3D Viewer is limited to viewing 3DMFs. It does not allow you to edit them. A far more capable freeware application is Geo3D by Stefan Huber which is a better viewer and adds some editing and animation features (www.topoi.ch). If you need a full-blown modelling, rendering and animation freeware package you may wish to consider Strata3D (www.strata.com/html/demos_updates.html). This is a stripped down version of what used to be Strata StudioPro. Finally, most commercial 3D applications edit 3DMFs.

Conclusion

This article showed you how to use AppleScript to take a raw 3D text file and convert it into the 3DMF file format, viewable by any QuickDraw 3D application. Part of the fun was that the technologies and applications that we used — QuickDraw 3D, the QuickDraw 3D Viewer, AppleScript and ScriptMaker — all come standard with your PowerMacintosh. You can of course push the boundaries further by trying to export 3D data from your favourite commercial database or spreadsheet application. Enjoy!

By Tim Monroe

The Atomic Café

Working With Atoms and Atom Containers

Introduction

In the past two *QuickTime Toolkit* articles, we've been concerned at least in part with *atoms*, the basic building-blocks of QuickTime movie files. Atoms are utterly simple in structure (a 4-byte length field, followed by a 4-byte type field, followed by some data), and this utter simplicity means that atoms can be used for a very wide range of tasks. Indeed, the atom-based structure used by QuickTime movie files is so general and so flexible that it has been adopted by the International Standards Organization (ISO) as the basis for the development of a unified digital media storage format for the emerging MPEG-4 specification.

In this article, we're going to continue investigating the basic structure of QuickTime files as sequences of atoms. You might recall that in the previous article we left some unfinished business lying around. Namely, we need to see how to replace an existing atom of a particular type instead of just adding a new atom of that type. It turns out that handling this task in the general case is reasonably difficult, since we can't safely move the movie data atom around in a movie file without doing a lot of work. For the present, we'll be content to see how to amend the QTInfo_MakeFilePreview function we developed last time so that there is at most one 'pnot' atom in a QuickTime movie file.

Because an atom can contain any kind of data whatsoever, it can contain data that consists of one or more other atoms. So, atoms can be arranged hierarchically. We'll take a few moments to consider the hierarchical arrangement of a movie atom (the main repository of bookkeeping information in a QuickTime movie file). Then we'll show how to put our atom-fusing powers to work to create a *shortcut movie file*, a QuickTime movie file that does nothing more than point to some other QuickTime movie file.

Once we've played with atoms for a while, we're going to shift gears rather abruptly to consider a second kind of atom-based structure, which we'll call an *atom container*. Atom containers are structures of type QTAtomContainer that are often used inside of QuickTime movie data atoms to store various kinds of information (for example, media samples). They were developed primarily to address some of the shortcomings of atoms. In particular, the Movie Toolbox provides an extensive API for working with atom containers; among other things, this API makes it easy to create and access data arranged hierarchically within atom containers.

We'll get some hands-on experience with atom containers in two ways. First, we'll see how to get and set the user's Internet connection speed preference, which is a piece of information that QuickTime stores internally and happily gives, in the form of an atom container, to anyone who asks. Second, we'll see how to add a *movie track* to a QuickTime movie. By using movie tracks, we can embed entire QuickTime movies inside of other QuickTime movies. The *movie media handler*, which manages movie tracks, is one of the most exciting new features of QuickTime 4.1. Once we understand how to work with atom containers, it'll be easy to add movie tracks to an existing QuickTime movie.

Before we begin, though, a word about terminology. As you've been warned, this article is going to discuss two different ways of organizing data, both of which are (for better or worse) called "atoms". The first kind of atom is the one that's used to define the basic structure of QuickTime movie files. The second kind of atom is the one that was introduced in QuickTime 2.1

Tim Monroe recently acquired 3 pet lizards (green anoles), not realizing that he'd need to feed them things like crickets, mealworms, and spiders. His house is now surprisingly spider-free, however. You can send your bugs to him at monroe@apple.com.

for storing data in some kinds of media samples (and for other tasks as well); these kinds of atoms are structures of type QTAtom that are stored inside of an atom container.

Some of Apple's QuickTime documentation refers to the first kind of atom as a *classic atom* (perhaps in the same spirit that one refers to a classic car: it's been around a while) and to the second kind of atom as a *QT atom* (drawing of course on the data type QTAtom). Some other documentation refers to the first kind of atom as a *chunk atom* (perhaps because it's just a chunk of data?). I'm not particularly happy with any of these terms, so I'm going to refer to the first kind of atom simply as an *atom* and to the second kind as an *atom container atom*. In other words, an atom container atom (of type QTAtom) is always found inside of an atom container (of type QTAtomContainer). Generally, here and in the future, the context will make it clear which kind of atom we're considering, so we can usually get by just talking about atoms.

File Previews: The Sequel

In the previous article ("The Informant" in *MacTech*, July 2000), we saw how to add a 'pnot' atom to a QuickTime movie file, to create a single-fork movie file with a file preview. (A *file preview* is the movie clip, image, text, or other data that appears in the preview pane of the file-opening dialog box displayed by a call to StandardGetFilePreview or NavGetFile.) Our strategy was simple: each time the user saves a movie, add a preview atom and (if necessary) a preview data atom to the QuickTime movie file. But we recognized that ultimately we would need to refine this strategy to avoid ending up with multiple preview atoms and preview data atoms. It's time to make some changes to our QTInfo application. In this section, we'll see how to upgrade QTInfo into a nearly-identical application, called QTInfoPlus, that handles file preview atoms correctly.

Removing Existing Previews

In fact, we can solve this little problem by adding a single line of code to the QTInfo_MakeFilePreview function. Immediately after determining that the file reference number passed to QTInfo_MakeFilePreview picks out a data file, we can execute this code:

```
QTInfo_RemoveAllPreviewsFromFile(theRefNum);
```

The QTInfo_RemoveAllPreviewsFromFile function looks through the specified open data file and removes any existing preview atoms (that is, atoms of type 'pnot') from that file. In addition, this function removes any preview data atoms referenced by those preview atoms, unless the preview data atoms are of type 'moov'. (We don't want to remove atoms of type 'moov', of course, since they contain essential information about the movie file.) QTInfo_RemoveAllPreviewsFromFile is defined in Listing 1.

Listing 1: Removing all preview atoms from a QuickTime movie file

QTInfo_RemoveAllPreviewsFromFile

```
OSErr QTInfo_RemoveAllPreviewsFromFile (short theRefNum)
{
  long       myAtomType = 0L;
  short      myAtomIndex = 0;
  short      myCount = 0;
  OSErr      myErr = noErr;

  // count the preview atoms in the file
  myCount = QTInfo_CountAtomsOfTypeInFile(theRefNum, 0L,
                        ShowFilePreviewComponentType);
  while (myCount > 0) {
    // get the preview data atom targeted by this preview atom
    myAtomType = ShowFilePreviewComponentType;
    myAtomIndex = myCount;
    myErr = QTInfo_FindPreviewAtomTarget(theRefNum,
                        &myAtomType, &myAtomIndex);

    // if the preview data atom is the last atom in the file, remove it
    // (unless it's a 'moov' atom)
    if (myErr == noErr)
      if (myAtomType != MovieAID)
        if (QTInfo_IsLastAtomInFile(theRefNum,
                        myAtomType, myAtomIndex))
          QTInfo_RemoveAtomFromFile(theRefNum,
                        myAtomType, myAtomIndex);

    // remove or free the preview atom
    if (QTInfo_IsLastAtomInFile(theRefNum,
                ShowFilePreviewComponentType, myCount))
      QTInfo_RemoveAtomFromFile(theRefNum,
                ShowFilePreviewComponentType, myCount);
    else
      QTInfo_FreeAtomInFile(theRefNum,
              ShowFilePreviewComponentType, myCount);
```

```
    // if the preview data atom still exists, remove or free it (unless it's a 'moov' atom)
    if (myErr == noErr)
      if (myAtomType != MovieAID)
        if (QTInfo_IsLastAtomInFile(theRefNum,
                                    myAtomType, myAtomIndex))
          QTInfo_RemoveAtomFromFile(theRefNum,
                                    myAtomType, myAtomIndex);
        else
          QTInfo_FreeAtomInFile(theRefNum,
                                myAtomType, myAtomIndex);

      myCount--;
    }

  return(myErr);
}
```

As you can see, QTInfo_RemoveAllPreviewsFromFile calls a handful of other functions defined by our application. These other functions do things like count the number of existing preview atoms, find the preview data atom that is the target of a preview atom, determine whether a given atom is the last atom in the file, and so forth.

QTInfo_RemoveAllPreviewsFromFile puts these functions to work like this: for each preview atom in the file (starting with the one nearest the end of the file), find the preview data atom that is referenced by the preview atom. If that target atom isn't a movie atom and it's the last atom in the file, remove it from the file. Then, if the preview atom is the last atom in the file, remove it as well. If the preview atom isn't the last atom in the file, then change it into a *free atom* (that is, an atom whose type is FreeAtomType). By changing the atom type, we're converting the preview atom into a block of unused space at its current location in the movie file. QuickTime simply ignores any atoms of type FreeAtomType that it encounters when reading through a movie file.

You might think that we could just remove a preview atom from the file and, if it isn't the last atom in the file, move any following atoms up in the file. This would avoid creating "islands" of unused space in our file, but it would be a dangerous thing to do. That's because some atoms in a QuickTime file reference data in other atoms by storing offsets from the beginning of the file. In general, we want to avoid moving atoms around in a QuickTime movie file. It's safer just to convert any unwanted atoms that are not at the end of the file into free atoms.

Once we've removed a preview atom (if it's the last atom in the file) or converted it into a free atom (if it isn't), we then look once again to see if the preview data atom is the last item in the file. This might happen if the preview data atom originally preceded the preview atom and the preview atom was the last atom in the file. If the preview data atom is now the last atom in the file, it's removed; otherwise, it's converted into a free atom.

The net result of all this is to remove any existing preview and preview data atoms from the file, either by truncating the file to exclude those atoms or by converting them into free atoms. At this point, QTInfo_MakeFilePreview can safely add a new preview atom and (if necessary) a preview data atom to that file. So, when all is said and done, the QuickTime movie file will end up with exactly one preview atom and one preview data atom. In the next few subsections, we'll consider how to define the various QTInfoPlus functions called by QTInfo_RemoveAllPreviewsFromFile.

Finding and Counting Atoms

The most fundamental thing we need to be able to do, when working with a file that's composed of atoms, is find an atom of a specific type and index in that file. For instance, we might need to find the first movie atom, or the third preview atom, or the third 'PICT' atom in the file. So we want to devise a function that takes an atom type and an index and then returns to us the position in the file at which that atom begins, if there is an atom of that type and index in the file. Otherwise, the function should return some error.

This task is reasonably straightforward. All we need to do is start at the beginning of the file (or at some other offset in the file specified by the caller) and inspect the type of the atom at that location. If the desired index is 1 and the desired atom type is the type of that atom, we're done: we've found the desired atom. Otherwise, we need to keep looking. We can find the next atom in the file by moving forward in the file by the size of the atom currently under consideration. We continue inspecting each atom and moving forward in the file until we find the atom of the specified type and index or until we reach the end of the file. Listing 2 defines the function QTInfo_FindAtomOfTypeAndIndexInFile, which is our basic atom-finding tool.

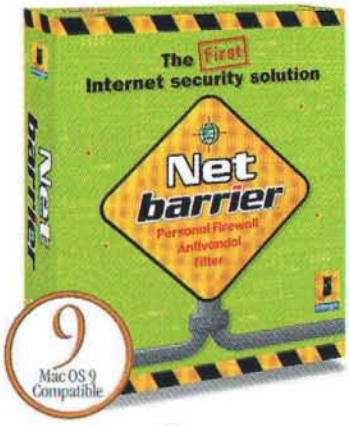
Listing 2: Finding an atom in a QuickTime movie file

QTInfo_FindAtomOfTypeAndIndexInFile

```
OSErr QTInfo_FindAtomOfTypeAndIndexInFile (short theRefNum,
        long *theOffset, long theAtomType, short theIndex,
          long *theDataSize, Ptr *theDataPtr)
{
  short         myIndex = 1;
  long          myFileSize;
  long          myFilePos = 0L;
  long          myAtomHeader[2];
  long          mySize = 0L;
  OSType        myType = 0L;
  Ptr           myDataPtr = NULL;
  Boolean       isAtomFound = false;
  OSErr         myErr = paramErr;

  if (theOffset == NULL)
    goto bail;

  if (QTInfo_IsRefNumOfResourceFork(theRefNum))
    goto bail;

  myFilePos = *theOffset;

  // get the total size of the file
  GetEOF(theRefNum, &myFileSize);

  while (!isAtomFound) {
    myErr = SetFPos(theRefNum, fsFromStart, myFilePos);
    if (myErr != noErr)
      goto bail;

    // read the atom header at the current file position
    mySize = sizeof(myAtomHeader);
    myErr = FSRead(theRefNum, &mySize, myAtomHeader);
    if (myErr != noErr)
      goto bail;
```

You think the Internet is safe. Think again...

NetBarrier. The first Internet security solution for Macintosh.



All Macs connected to the Internet (dialup, DSL, cable-modem) are exposed to hackers. Whether you are a home user or a professional user, your data interests them. That's why you need a security solution that only NetBarrier can provide.

Personal Firewall

NetBarrier protects and monitors all incoming and outgoing data. A customized mode allows you to create your own defense rules, thereby offering the most secure level of protection.

Antivandal

NetBarrier blocks all attempts to break into your Mac, detects wrong passwords and logs vandal attacks for complete protection. Moreover, it has an alarm to inform you of every intrusion attempt.

Internet filter

NetBarrier analyzes data as it leaves your computer and prevents unauthorized exporting of private information such as credit card numbers, passwords, sensitive data and more...

Available @ Developer DEPOT
<http://www.devdepot.com>
Toll Free: **877-DEPOT-NOW**
Outside U.S./Canada: **805/494-9797**

www.intego.com

intego

MacNN ★★★★ Macworld Macworld 4/5 Apple Wizards

```
    mySize = EndianU32_BtoN(myAtomHeader[0]);
    myType = EndianU32_BtoN(myAtomHeader[1]);

    if ((myIndex == theIndex) &&
            ((theAtomType == myType)
                || (theAtomType == kQTInfoAnyAtomType))) {
      // we found an atom of the specified type and index;
      // return the atom if the caller wants it
      if (theDataPtr != NULL) {
        myDataPtr = NewPtrClear(mySize);
        if (myDataPtr == NULL) {
          myErr = MemError();
          goto bail;
        }

        // back up to the beginning of the atom
        myErr = SetFPos(theRefNum, fsFromStart, myFilePos);
        if (myErr != noErr)
          goto bail;

        myErr = FSRead(theRefNum, &mySize, myDataPtr);
        if (myErr != noErr)
          goto bail;
      }

      isAtomFound = true;
    } else {
    // we haven't found an atom of the specified type and index; keep on looking
      myFilePos += mySize;
      if ((theAtomType == myType) ||
                        (theAtomType == kQTInfoAnyAtomType))
        myIndex++;

      // make sure we're moving forward in the file, but not too far...
      if ((mySize <= 0) ||
        (myFilePos > (myFileSize - sizeof(myAtomHeader)))) {
        myErr = cannotFindAtomErr;
        goto bail;
      }
    }
  } // while (!isAtomFound)

  // if we got to here, we found the correct atom
  if (theOffset != NULL)
    *theOffset = myFilePos;

  if (theDataPtr != NULL)
    *theDataPtr = myDataPtr;

  if (theDataSize != NULL)
    *theDataSize = mySize;

bail:
  if (myErr != noErr)
    if (myDataPtr != NULL)
      DisposePtr(myDataPtr);

  return(myErr);
}
```

QTInfo_FindAtomOfTypeAndIndexInFile returns to its caller the offset within the file of the beginning of the atom of the desired type and index. In addition, if the caller passes in non-NULL values in the theDataPtr or theDataSize parameters, QTInfo_FindAtomOfTypeAndIndexInFile returns a copy of the entire atom (including the atom header) or the atom size to the caller. The returned offset, data, and atom size can be used for a variety of purposes. For instance, Listing 3 defines the QTInfo_CountAtomsOfTypeInFile function, which counts the number of atoms of a specific type in a file.

Listing 3: Counting the atoms in a QuickTime movie file

QTInfo_CountAtomsOfTypeInFile

```
short QTInfo_CountAtomsOfTypeInFile (short theRefNum,
                              long theOffset, long theAtomType)
{
  short         myIndex = 0;
  long          myFilePos = theOffset;
  long          myAtomSize = 0L;
  OSErr         myErr = noErr;

  if (QTInfo_IsRefNumOfResourceFork(theRefNum))
    return(myIndex);

  while (myErr == noErr) {
    myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
            &myFilePos, theAtomType, 1, &myAtomSize, NULL);
    if (myErr == noErr)
      myIndex++;
    myFilePos += myAtomSize;

    // avoid an infinite loop...
    if (myAtomSize <= 0)
      break;
  }

  return(myIndex);
}
```

QTInfo_CountAtomsOfTypeInFile uses the offset and atom size returned by QTInfo_FindAtomOfTypeAndIndexInFile to walk through the file looking for atoms of the specified type.

Similarly, it's easy to use QTInfo_FindAtomOfTypeAndIndexInFile to determine whether a particular atom is the last atom in a file. We simply call QTInfo_FindAtomOfTypeAndIndexInFile to get the offset in the file of the given atom and then call it again to see if there are any atoms of any kind following that atom. Listing 4 defines a function that does precisely this.

Listing 4: Determining whether an atom is the last atom in a file

QTInfo_IsLastAtomInFile

```
Boolean QTInfo_IsLastAtomInFile (short theRefNum,
                          long theAtomType, short theIndex)
{
  Boolean       isLastAtom = false;
  long          myOffset = 0L;
  long          myAtomSize = 0L;
  OSErr         myErr = noErr;

  // find the offset and size of the atom of the specified type and index in the file
  myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
      &myOffset, theAtomType, theIndex, &myAtomSize, NULL);
  if (myErr == noErr) {
    // look for an atom of any type following that atom
    myOffset += myAtomSize;
    myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
            &myOffset, kQTInfoAnyAtomType, 1, NULL, NULL);
    if (myErr != noErr)
      isLastAtom = true;
  }

  return(isLastAtom);
}
```

Finding the Preview Data Atom

Given a preview atom, we sometimes need to know which other atom is the target of that atom. In other words, we want to find the atom that we've been calling the preview data atom — the atom that contains the data for the file preview. This is fairly easy:

we just need to read the data in the preview atom, which has the structure of a PreviewResourceRecord record. Listing 5 defines the QTInfo_FindPreviewAtomTarget function, which does this.

Listing 5: Finding the target of a preview atom

QTInfo_FindPreviewAtomTarget

```
OSErr QTInfo_FindPreviewAtomTarget (short theRefNum,
                        long *theAtomType, short *theIndex)
{
  long                    myOffset = 0L;
  PreviewResourceRecord   myPNOTRecord;
  long                    mySize;
  OSErr                   myErr = noErr;

  if ((theAtomType == NULL) || (theIndex == NULL))
    return(paramErr);

  // find the offset of the atom of the specified type and index in the file
  myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
          &myOffset, *theAtomType, *theIndex, NULL, NULL);
  if (myErr == noErr) {
    // set the file mark to the beginning of the atom data
    myErr = SetFPos(theRefNum, fsFromStart,
                            myOffset + (2 * sizeof(long)));
    if (myErr == noErr) {
      // read the atom data
      mySize = sizeof(myPNOTRecord);
      myErr = FSRead(theRefNum, &mySize, &myPNOTRecord);
      if (myErr == noErr) {
        *theAtomType = EndianU32_BtoN(myPNOTRecord.resType);
        *theIndex = EndianS16_BtoN(myPNOTRecord.resID);
      }
    }
  }

  return(myErr);
}
```

QTInfo_FindPreviewAtomTarget calls QTInfo_FindAtomOfTypeAndIndexInFile to find the location of the atom whose type and index are passed to it in the theAtomType and theIndex parameters. Then it advances the file mark to the beginning of the atom data and reads the atom data into myPNOTRecord. The type and index of the preview data atom (suitably converted from big-endian to native-endian form) are then returned to the caller.

Removing and Freeing Atoms

It's quite easy to remove an atom from a file or convert it into a free atom. Listing 6 shows how we define the QTInfo_RemoveAtomFromFile function, which removes an atom from a file by truncating the file at the beginning of the atom (by calling SetEOF). Note that any atoms that follow the specified atom are also removed from the file. To avoid any problems, we should always call QTInfo_IsLastAtomInFile to make sure that the atom to be removed from the file is the last atom in the file.

Listing 6: Removing an atom from a file

QTInfo_RemoveAtomFromFile

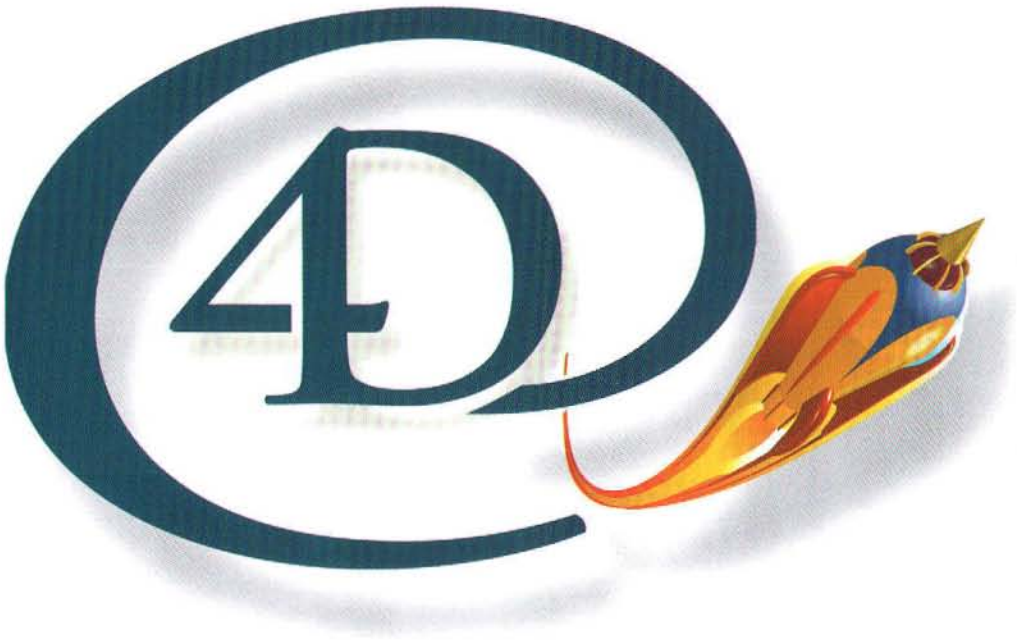
```
OSErr QTInfo_RemoveAtomFromFile (short theRefNum,
                        long theAtomType, short theIndex)
{
  long          myOffset = 0L;
  long          myAtomSize = 0L;
  OSErr         myErr = noErr;

  // find the offset of the atom of the specified type and index in the file
  myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
      &myOffset, theAtomType, theIndex, &myAtomSize, NULL);
  if (myErr == noErr)
    myErr = SetEOF(theRefNum, myOffset);

  return(myErr);
}
```

It's almost as easy to convert an atom into a free atom. All we need to do is position the file mark to the beginning of the type field in the atom header and write the value 'free' into that field. Listing 7 shows how we do this.

Listing 7: Freeing an atom in a file

QTInfo_FreeAtomInFile

```
OSErr QTInfo_FreeAtomInFile (short theRefNum,
                        long theAtomType, short theIndex)
{
  OSType        myType = EndianU32_NtoB(FreeAtomType);
  long          mySize = sizeof(myType);
  long          myOffset = 0L;
  OSErr         myErr = noErr;

  // find the offset of the atom of the specified type and index in the file
  myErr = QTInfo_FindAtomOfTypeAndIndexInFile(theRefNum,
          &myOffset, theAtomType, theIndex, NULL, NULL);
  if (myErr == noErr) {
    // change the atom type to 'free'
    myErr = SetFPos(theRefNum, fsFromStart,
                                myOffset + sizeof(long));
    if (myErr == noErr)
      myErr = FSWrite(theRefNum, &mySize, &myType);
  }

  return(myErr);
}
```


4D

@ 4D speed.

It's the speed at which business moves and the development pace you as a web professional understand all too well. Your development and hosting tools are your livelihood requiring absolute performance and delivery. Here at 4D, Inc. we acknowledge that fundamental necessity by offering the most powerful and intuitive Web Development Tools and Internet Servers on the market - 4D and the WebSTAR Server Suite.

4D Version 6.5 offers the most compelling web application development platform available. Having previously set the standard for seamless, cross-platform database development, 4D delivers a critically-acclaimed, multi-threaded and integrated web engine for web database Publishing. Solutions you build today carry forward with true scalability, integration and backwards compatibility. Simply put, your development effort is preserved freeing you to perform.

The WebSTAR Server Suite is the web-serving de-facto standard for Mac delivering the fastest, most comprehensive suite of Internet Servers on the planet. The award-winning administration interface puts you on the web in a fraction of the time it takes with other products. WebSTAR's security features are unequalled which is why WebSTAR is used by the U.S. military. And when it comes to speed, let's just say – WebSTAR rocks.

The speed at which you create solutions and host sites is what 4D and WebSTAR are all about. Slam your box with 4D and WebSTAR and you'll quickly find yourself @ 4D speed. Free demos are a URL away. Hey, how about downloading or calling 1.800.881.3466 for a Demo CD right now?

www.4D.com | **www.WebSTAR.com**

So far, then, we've managed to define a handful of utility functions that allow us to find atoms in files, get the sizes of those atoms, remove atoms from files, count the number of atoms of a specific type in a file, and so forth. These are precisely the functions called by the QTInfo_RemoveAllPreviewsFromFile function, which we're using to make sure that any QuickTime movie file we create or edit has at most one preview resource. It would be easy (and fun) to define an entire library of atom utilities, but we'll have to restrain our programming urges here. We've got other work to do.

Shortcut Movie Files

I mentioned earlier that an atom can contain any kind of data, and in particular it can contain other atoms. That is to say, atoms can be arranged hierarchically. Up to now, however, we've worked with a QuickTime movie file as a mere concatenation of atoms. We've looked at the data of several of those atoms, but we haven't yet met any atoms that contain other atoms. It's time for that to change.

A good example of an atom that contains other atoms is the movie atom itself. A typical movie atom contains a track atom (of type TrackAID) for each track in the QuickTime movie, along with other atoms that contain the movie metadata. A movie atom can also contain a movie user data atom (of type UserDataAID), which contains the movie user data. A track atom, in turn, contains other atoms that define the track characteristics and that point to the media data. And so on, as deep as is necessary to completely characterize a movie and its data.

An atom that contains no other atoms is called a *leaf atom*. A leaf atom may or may not actually contain any data. Typically a leaf atom does contain data, but it's possible that the very presence in the file of the atom has significance. In that case, the leaf atom consists solely of the 8-byte atom header. An atom that contains one or more other atoms is called a *container atom*. A movie atom is a container atom. By contrast, a preview atom is a leaf atom, since it contains data but no other atoms. (A preview atom points to or references another atom, but it does not contain it.)

Let's build a container atom. Now, it's beyond our current capabilities to build a typical movie atom, with all its complicated subatoms and subsubatoms. But there is a kind of QuickTime movie file that consists entirely of a movie atom and which is simple enough for us to build; this kind of file is called a *shortcut movie file*. A shortcut movie file is a movie file that picks out a single other movie file. It's rather like an alias file in the Macintosh file system or a shortcut on Windows. Opening a shortcut movie file using the Movie Toolbox function OpenMovieFile causes QuickTime to look for the file that is referred to by the shortcut movie file; if that target file can be found, then OpenMovieFile opens it and returns to the caller a file reference number for that target file.

Shortcut movie files provide a cross-platform mechanism for referring to QuickTime movie files. They can be useful in all the same ways that alias files (on Mac) or shortcuts (on Windows) can be useful, at least when working with QuickTime movies. For instance, a web page might contain an embedded URL to a shortcut movie file. If the webmaster wants to update the movie displayed in the web page, he or she needs only to create a new shortcut movie file that refers to the updated movie and then put the new shortcut movie file in the location occupied by the previous shortcut movie file. In this way, the contents of the web page can be changed without altering the actual HTML tags of the page.

QuickTime has supported shortcut movie files since version 3.0. QuickTime version 4.0 introduced the Movie Toolbox function CreateShortcutMovieFile, which we can use to create a shortcut movie file. But the structure of shortcut movie files is so simple that we can build them ourselves. Let's see how to do this.

The format of shortcut movie files is (to my knowledge) currently undocumented, but it's quite simple: the shortcut movie file consists entirely of a single movie atom, which in turn contains a movie data reference alias atom (of type MovieDataRefAliasAID). This atom contains a single data reference atom (of type DataRefAID). Finally, the data reference atom is a leaf atom that contains the type of the data reference followed immediately by the data reference itself. **Figure 1** shows the general structure of a shortcut movie file.

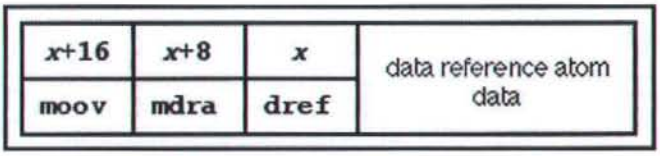


Figure 1. *The structure of a shortcut movie file.*

It's easy enough to create a file that has this structure. The only thing we don't yet know is what a data reference is, to put into the data reference atom. In the next *QuickTime Toolkit* article, we'll take a long look at data references; for the moment, we'll just suppose that a suitable data reference is passed to us. (To look ahead a bit, a data reference is a handle to some data that picks out a movie file or other file. For instance, a URL data reference is a handle to the NULL-terminated string of characters in the URL.) Listing 8 shows the function QTShortCut_CreateShortcutMovieFile that takes that data reference and data reference type and then builds a shortcut movie file.

Listing 8: Creating a shortcut movie file

QTShortCut_CreateShortcutMovieFile

```
OSErr QTShortCut_CreateShortcutMovieFile (Handle theDataRef,
            OSType theDataRefType, FSSpecPtr theFSSpecPtr)
{
  long        myVersion = 0L;
  OSErr       myErr = noErr;

  myErr = Gestalt(gestaltQuickTime, &myVersion);
  if (myErr != noErr)
    goto bail;
```

```
    if (((myVersion >> 16) & 0xffff) >= 0x0400) {
      // we're running under QuickTime 4.0 or greater
      myErr = CreateShortcutMovieFile(theFSSpecPtr,
                        sigMoviePlayer,
                        smCurrentScript,
                        createMovieFileDeleteCurFile |
                            createMovieFileDontCreateResFile,
                        theDataRef,
                        theDataRefType);

    } else {
      // we're running under a version of QuickTime prior to 4.0
      OSType              myDataRefType;
      unsigned long       myAtomHeaderSize;
      Ptr                 myData = NULL;
      Handle              myAtom = NULL;

      // create the atom data that goes into a data reference atom (we will create this
      // atom's header when we create the movie atom that contains it); the atom data is
      // the data reference type followed by the data reference itself
      myDataRefType = EndianU32_NtoB(theDataRefType);
      myAtomHeaderSize = 2 * sizeof(long);

      // allocate a data block and copy the data reference type and data reference into it
      myData = NewPtrClear(sizeof(OSType) +
                           GetHandleSize(theDataRef));
      if (myData == NULL)
        goto bail;

      BlockMove(&myDataRefType, myData, sizeof(OSType));
      BlockMove(*theDataRef, (Ptr)(myData + sizeof(OSType)),
                                   GetHandleSize(theDataRef));

      // create a handle to contain the size and type fields of the movie atom, as well as
      // the size and type fields of the movie data reference alias atom contained in it
      // and of the data reference atom contained in the movie data reference alias atom
      myAtom = NewHandleClear(3 * myAtomHeaderSize);
      if (myAtom == NULL)
        goto bail;

      // fill in the size and type fields of the three atoms
      *((long *)(*myAtom + 0x00)) = EndianU32_NtoB((3 *
                        myAtomHeaderSize) + GetPtrSize(myData));
      *((long *)(*myAtom + 0x04)) = EndianU32_NtoB(MovieAID);
      *((long *)(*myAtom + 0x08)) = EndianU32_NtoB((2 *
                        myAtomHeaderSize) + GetPtrSize(myData));
      *((long *)(*myAtom + 0x0C)) =
                       EndianU32_NtoB(MovieDataRefAliasAID);
      *((long *)(*myAtom + 0x10)) = EndianU32_NtoB((1 *
                        myAtomHeaderSize) + GetPtrSize(myData));
    *((long *)(*myAtom + 0x14)) = EndianU32_NtoB(DataRefAID);

      // concatenate the data in myData onto the end of the movie atom
      myErr = PtrAndHand(myData, myAtom, GetPtrSize(myData));
      if (myErr != noErr)
        goto bail;

      // create the shortcut movie file
      myErr = QTShortCut_WriteHandleToFile(myAtom,
                                                 theFSSpecPtr);
bail:
      if (myData != NULL)
        DisposePtr(myData);

      if (myAtom!= NULL)
        DisposeHandle(myAtom);
    }

    return(myErr);
}
```

Our function QTShortCut_CreateShortcutMovieFile calls the Movie Toolbox function CreateShortcutMovieFile if it's available; otherwise it creates the movie atom itself, by building up three consecutive atom headers and then appending the data reference type and data onto the end of those atom headers. It writes the entire movie atom into the specified file by calling the QTShortCut_WriteHandleToFile function. (We've already encountered a version of this function, called QTDX_WriteHandleToFile; see "In and Out" in *MacTech*, May 2000.)

Atom Containers

QuickTime version 2.1 introduced a new way to store information that greatly facilitates creating hierarchical collections of data and retrieving data from those collections. The basic ideas are very simple: at the root of the data hierarchy is an object called an *atom container*. Inside of an atom container are other objects, called *atoms*. (If we need to distinguish these atoms from the ones we've been considering up to now, we'll call them *atom container atoms*.) An atom can contain other atoms, in which case it is a *parent atom*. The atoms that are contained within a parent atom are called its *child atoms*. If an atom contains only data (and no other atoms), it is a *leaf atom*. The data in a leaf atom is always in big-endian format. (Well, *almost* always; we'll encounter an exception to this rule in a little while.)

An atom has an atom type and an atom ID. A parent atom can contain any number of children of any type and ID. The only restriction is that, for a given parent, no two children can have the same type and ID. So we can uniquely identify an atom by specifying its parent, its type, and its ID. (For an atom that is contained directly in the atom container, the atom container is considered to be the atom's parent; the special constant kParentAtomIsContainer is used to signal this fact.) We can also identify a particular atom by specifying its parent, its

type, and an index of atoms of that type in that parent. (QuickTime supports yet a third method of identifying atoms, using the atom's position in the atom container, called its *offset*; we won't consider this way of identifying atoms here.)

Let's consider a few examples. Figure 2 shows a very simple collection of data, where the atom container has just two children, each of which holds a long integer that represents the length (in millimeters) of a lizard. These leaf atoms both have the atom type 'lzln'.

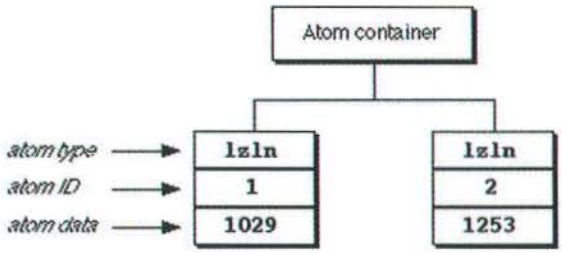


Figure 2. *A simple atom container.*

Figure 3 shows a more complicated arrangement of data. In this case, the root atom container contains two parent atoms, both of type 'ldat' (for "lizard data"). Within each 'ldat' atom are two children, which have different types. The atom of type 'lzln' contains a long integer (as in **Figure 2**) and the atom of type 'lnam' (for "lizard name") contains a string of characters. (This is neither a C string nor a Pascal string; it's just the characters themselves.) Notice that both 'lzln' atoms have an atom ID of 1; this is okay, since those atoms have different parents.

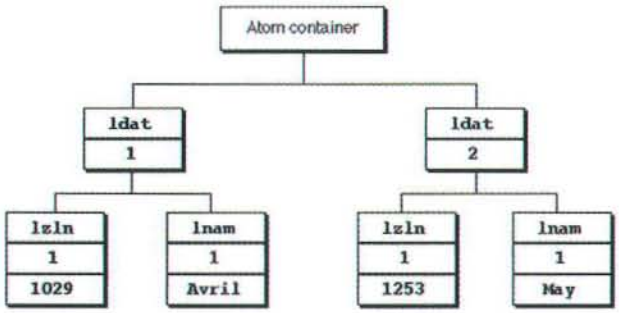


Figure 3. *A more complex atom container.*

Atom containers can be vastly more complicated than the ones shown in **Figures 2 and 3**, and they don't have to exhibit the kind of nice symmetry we see there. On the other hand, some real-life atom containers are just that simple. But no matter how complicated they are, we'll use the same functions to build atoms and atom containers and to retrieve data from atom containers. Let's see how to accomplish these tasks.

Creating Atom Containers

The file Movies.h defines these types for working with atoms and atom containers:

```
typedef Handle              QTAtomContainer;
typedef long                QTAtom;
typedef long                QTAtomType;
typedef long                QTAtomID;
```

Notice that an atom container is just a handle to some data (structured in a specific way, to be sure). This means that we can determine the size of an atom container by using the function GetHandleSize. That's about as much as we need to know about the way an atom container is stored in memory. The actual structure of an atom container is publicly documented, but thankfully we will not need to learn anything about that structure. The Movie Toolbox provides all the functions we'll need in order to create and use atoms and atom containers.

We create an atom container by calling the QTNewAtomContainer function, like this:

```
QTAtomContainer      myAtomContainer = NULL;
myErr = QTNewAtomContainer(&myAtomContainer);
```

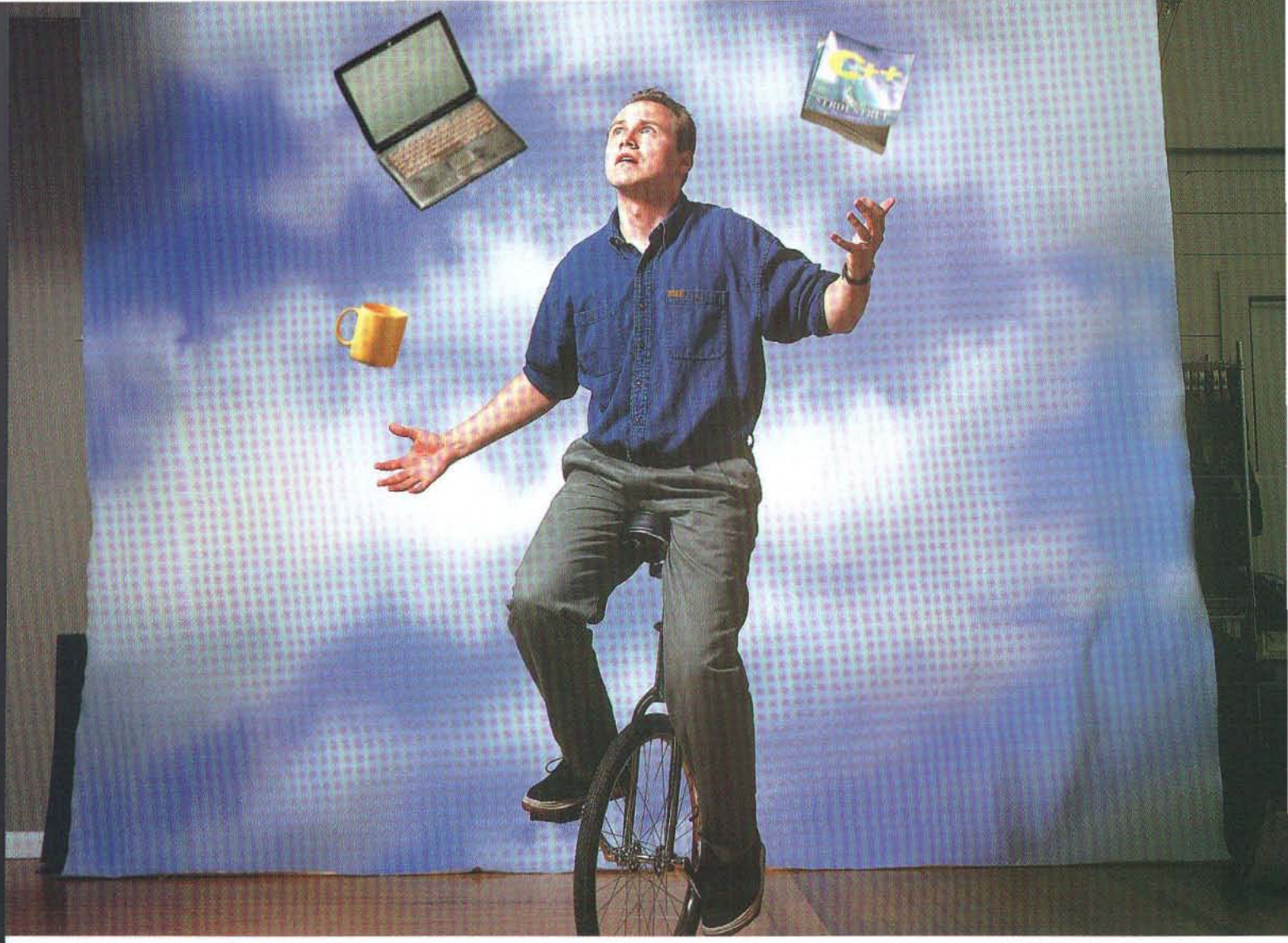
If QTNewAtomContainer completes successfully, then the value of the variable myAtomContainer is a new, empty atom container. We can then add atoms to that container by calling QTInsertChild. For instance, to add the two children shown in **Figure 2** to this atom container, we could execute this code:

```
myLong = EndianU32_NtoB(1029);
myErr = QTInsertChild( myAtomContainer,
                       kParentAtomIsContainer,
                           kLizardLength, 1, 0,
                           sizeof(myLong), &myLong, NULL);
myLong = EndianU32_NtoB(1253);
myErr = QTInsertChild( myAtomContainer,
                       kParentAtomIsContainer,
                           kLizardLength, 2, 0,
                           sizeof(myLong), &myLong, NULL);
```

The second parameter to the QTInsertChild function specifies the parent atom of the child we're inserting. Here, you'll notice, we're using the constant kParentAtomIsContainer to indicate that the parent atom is the atom container itself. The third and fourth parameters specify the type and ID of the new atom. The fifth parameter is the desired index of the new atom within the parent atom; we don't care about the index here, so we pass the value 0 to indicate that the new atom is to be inserted as the last child of the specified type in the parent atom.

The sixth and seventh parameters to QTInsertChild specify the number of bytes of data to be added to the atom, along with a pointer to the atom data itself. The last parameter is a pointer to a variable of type QTAtom, in which QTInsertChild will return to us an identifier for the new atom; we don't need that information here, so we pass NULL in that parameter.

We can create a hierarchy within an atom container by inserting parent atoms and then adding some children to those parents. We also call QTInsertChild to insert a parent atom, but



Programming Doesn't Have To Be This Difficult. It's REALbasic!

NEW Version! REALbasic 2.1* FREE Update!

REALbasic is the award-winning, visual, object-oriented BASIC development environment for the Macintosh.

Use REALbasic's visual interface builder and platform-independent language to build native, compiled—not interpreted—professional quality applications in a fraction of the time it would take in C/C++. Because our language is platform-independent you only need to write a single set of code to create applications for both Macintosh and Windows. Leverage your C/C++ experience to extend REALbasic's capabilities using shared libraries, Mac OS Toolbox and Win32 API calls or by writing cross-platform REALbasic plug-ins.

Create prototypes, Internet applications, database front-ends, even games with REALbasic! Its OOP language employs everything you'd expect from a modern development environment—methods, properties, classes, subclassing, inheritance, constructors and destructors, virtual methods, and more. The IDE supports multi-threading, extensibility, TCP/IP controls, plus multimedia and QuickTime tools, and support for standards such as SQL, ODBC, AppleScript, and XCMDs. You can even import Visual Basic forms and modules.

Go to www.realbasic.com NOW to download a FREE trial version or call 512.263.1233.

REALbasic®

MacPower 2000 Products of The Year

MacHome

Macworld

Apple Design Award

2000 Runner-Up - Best Macintosh User Experience

Mac Fan MVP '99 MOST VALUABLE PRODUCT

*Free update for all owners of REALbasic 2.0 and above. REALbasic and the REALbasic logo are trademarks of REAL Software, Inc. Apple and the Apple logo are trademarks of Apple Computer, Inc., registered in the U.S., used with permission. All other trademarks are the property of their respective owners.

we do not need to specify any data or data size; instead, we specify a variable of type QTAtom in which the identifier of the new parent atom is returned to us. Here's an example:

```
QTAtom                  myLizardAtom;
myErr = QTInsertChild( myAtomContainer,
                        kParentAtomIsContainer,
                            kLizardData, 1, 1, 0, NULL,
                            &myLizardAtom);
```

Then we can insert a child atom into this parent atom, like this:

```
myErr = QTInsertChild( myAtomContainer,
                        myLizardAtom,
                            kLizardName, 1, 1,
                            strlen(theLizardName),
                            theLizardName, NULL);
```

Note that the second parameter here is the parent atom that we just created. If we insert another atom (this time of type 'lzln') into the parent and then repeat the whole process for the second lizard, we'd have the atom structure shown in **Figure 3**.

Finding Atoms in Atom Containers

If we are given an atom container, it's almost as easy to get data out of it as it is to put data into it. First we need to find the atom whose data we want. The standard way to do this is to start at the top of the hierarchy and gradually descend until we find the parent of the desired atom. Then we can get an atom identifier of the target atom by calling the QTFindChildByID function. For example, if myLizardAtom is the parent atom for the atoms that hold the data about our lizard Avril, then we can get the name atom by executing this code:

```
myNameAtom = QTFindChildByID(myAtomContainer,
myLizardAtom,
                        kLizardName, 1, NULL);
```

QTFindChildByID actually inspects both the type and ID passed to it (not just the ID, as the name might suggest).

The Movie Toolbox provides a number of other functions that are useful for finding specific atoms, including QTCountChildrenOfType, QTFindChildByIndex, QTGetNextChildType, and QTNextChildAnyType.

Getting Atom Data

Once we've found a leaf atom, we can get the data from that atom in several ways. If we want a copy of the atom data that will persist even after we've disposed of the atom container, we can call QTCopyAtomDataToHandle or QTCopyAtomDataToPtr, passing in a handle or pointer to a block of memory that's big enough to hold the leaf atom data. If, on the other hand, we just want to look at the atom data and don't need a copy of it, we can call the QTGetAtomDataPtr function, which returns a pointer to the actual leaf atom data. If you plan to make calls that might move memory, then you should call QTLockContainer before calling QTGetAtomDataPtr; then call QTUnlockContainer when you are done with the data pointer.

If we want to retrieve our lizard's name, we could make this call:

```
QTGetAtomDataPtr(myAtomContainer, myNameAtom, &myNameSize,
                                                &myNameData);
```

If QTGetAtomDataPtr completes successfully, then myNameData points to the string of characters that make up the name, and myNameSize contains the size of that name.

Internet Connection Speed

For our first real-life encounter with atom containers, let's consider how to get and set the user's Internet connection speed preference. The user can set a preference in the Connection Speed panel of the QuickTime™ Settings control panel, shown in **Figure 4**.

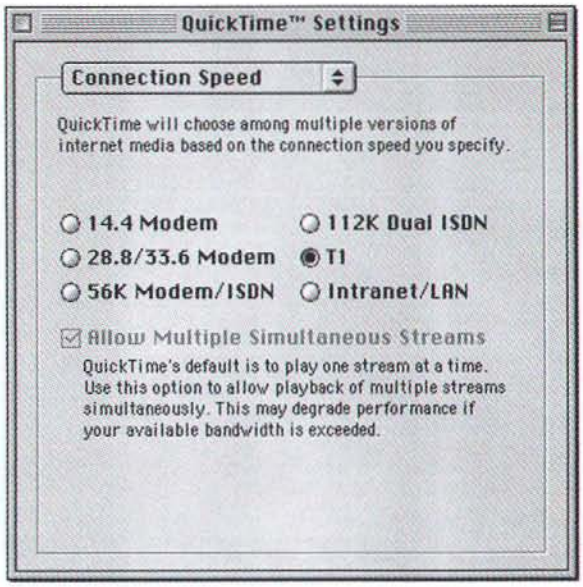


Figure 4. *The Connection Speed panel.*

QuickTime uses this setting for various purposes. For instance, if a user wants to play an alternate data rate movie file located on a remote server, QuickTime uses this connection speed to select the correct target movie. (An *alternate data rate movie file* is a movie file that references other movies, each tailored for downloading across a connection of a certain speed.)

We can retrieve the user's current QuickTime preferences by calling the GetQuickTimePreference function, like this:

```
myErr = GetQuickTimePreference(ConnectionSpeedPrefsType,
                                &myPrefsContainer);
```

The first parameter specifies the kind of preference we wish to retrieve, and the second parameter is the address of an atom container in which the requested preference data is returned. It's up to us to dispose of that atom container when we are done reading data from it. In the present case, when we retrieve the Internet connection speed, the atom container contains an

atom of type ConnectionSpeedPrefsType whose data is structured as a record of type ConnectionSpeedPrefsRecord, defined like this:

```
struct ConnectionSpeedPrefsRecord {
  long                    connectionSpeed;
};
```

This record contains a single field that indicates the number of bytes per second that the user's Internet connection can support. The file MoviesFormat.h defines a set of common values:

```
enum {
  kDataRate144ModemRate       = 1400,
  kDataRate288ModemRate       = 2800,
  kDataRateISDNRate             = 5600,
  kDataRateDualISDNRate       = 11200,
  kDataRateT1Rate               = 150000L,
  kDataRateInfiniteRate       = 0x7FFFFFFF
};
```

Once we've received an atom container from GetQuickTimePreference, we can use the QTFindChildByID function to find the child atom of type ConnectionSpeedPrefsType. Then we get the atom data by calling the QTGetAtomDataPtr function. Finally, we can read the value stored in the connectionSpeed field, to find the current connection speed preference. We'll return this value as the function result, whether or not it's one of the predefined common values. If any error occurs, however, we'll return the value kDataRate288ModemRate, which is a reasonable default. Listing 9 shows the complete function QTUtils_GetUsersConnectionSpeed.

Listing 9: Getting the user's Internet connection speed preference

QTUtils_GetUsersConnectionSpeed

```
long QTUtils_GetUsersConnectionSpeed (void)
{
  QTAtomContainer                 myPrefsContainer = NULL;
  QTAtom                          myPrefsAtom = 0;
  ConnectionSpeedPrefsRecord      myPrefsRec;
  long                            myDataSize = 0L;
  long                            mySpeed =
                                  kDataRate288ModemRate;
  Ptr                             myAtomData = NULL;
  OSErr                           myErr = noErr;

  myErr = GetQuickTimePreference(ConnectionSpeedPrefsType,
                                 &myPrefsContainer);
  if (myErr == noErr) {
    // find the atom of the desired type
    myPrefsAtom = QTFindChildByID(myPrefsContainer,
                    kParentAtomIsContainer,
                    ConnectionSpeedPrefsType, 1, NULL);
    if (myPrefsAtom != 0) {
      // read the data contained in that atom and verify that the data is of the
      // size we are expecting
      QTGetAtomDataPtr(myPrefsContainer, myPrefsAtom,
                                 &myDataSize, &myAtomData);
      if (myDataSize == sizeof(ConnectionSpeedPrefsRecord)) {
        // read the connection speed
        myPrefsRec =
                *(ConnectionSpeedPrefsRecord *)myAtomData;
        mySpeed = myPrefsRec.connectionSpeed;
      }
    }
    QTDisposeAtomContainer(myPrefsContainer);
  }

  return(mySpeed);
}
```

Note that we haven't performed any endian-swapping on the value we read from the connection speed preferences record. That's because the data in this particular atom container is stored in its native-endian format. This is an exception to the general rule that data in atom containers is big-endian. A user's QuickTime preferences are not designed to be moved from machine to machine, so there is no need to enforce big-endian byte ordering.

Listing 10 shows how we can set a user's Internet connection speed. In general, we should let the user decide the connection speed preference, but it can sometimes be useful to do this programmatically.

Listing 10: Setting the user's Internet connection speed preference

QTUtils_SetUsersConnectionSpeed

```
OSErr QTUtils_SetUsersConnectionSpeed (long theSpeed)
{
  QTAtomContainer                 myPrefsContainer = NULL;
  ConnectionSpeedPrefsRecord      myPrefsRec;
  OSErr                           myErr = noErr;

  myErr = QTNewAtomContainer(&myPrefsContainer);
  if (myErr == noErr) {
    myPrefsRec.connectionSpeed = theSpeed;
```

```
    myErr = QTInsertChild(myPrefsContainer,
                    kParentAtomIsContainer,
                    ConnectionSpeedPrefsType, 1, 0,
                    sizeof(ConnectionSpeedPrefsRecord),
                    &myPrefsRec, NULL);
    if (myErr == noErr)
      myErr = SetQuickTimePreference(
              ConnectionSpeedPrefsType, myPrefsContainer);

    QTDisposeAtomContainer(myPrefsContainer);
  }

  return(myErr);
}
```

QTUtils_SetUsersConnectionSpeed creates a new atom container, inserts a single child atom into the container that holds the desired speed, and then passes that container to the SetQuickTimePreference function. Once SetQuickTimePreference returns, we can safely call QTDisposeAtomContainer to dispose of the atom container we created.

Movie Tracks

You might recall that in an earlier article ("Opening The Toolbox" in *MacTech*, March 2000), we saw how to write an application that plays one QuickTime movie inside of another QuickTime movie. The embedded movie (what we then called the "picture-in-picture movie") could have looping characteristics different from those of the main movie. For example, the embedded movie could keep looping over and over while the main movie played though once and then stopped. And, in theory, the embedded movie could play at twice its normal speed while the main movie played at, say, half its normal speed. (We didn't actually provide this alternate speed capability, but it would be easy enough to add.)

The only drawback to this was that we needed the special playback application QTMooVToolbox to make it all happen. Wouldn't it be nice if we could create movie files with these capabilities, so that they would play back using any QuickTime-savvy application? This is precisely what's offered in QuickTime 4.1 with the introduction of *movie tracks* managed by the movie media handler. By adding movie tracks to an existing QuickTime movie, we can effectively embed an entire QuickTime movie into that movie. (This capability is sometimes called the *movie-in-movie* capability; the embedded movie is also called the *child movie*, while the main movie is also called the *parent movie*.) **Figure 5** shows one movie embedded within another movie using a movie track.

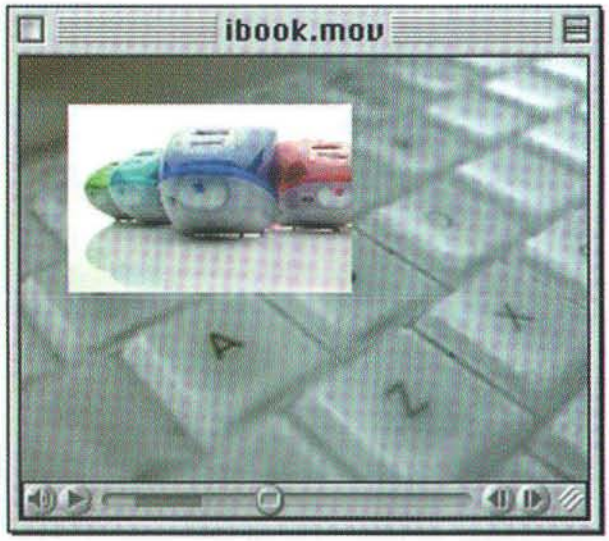


Figure 5. *A child movie inside of a parent movie.*

Remember that the looping characteristics and playback rate of a movie are associated with the movie's time base. Prior to QuickTime 4.1, it was possible to create movies with overlaid video tracks, but all the tracks in the movie shared the same time base. The time base of the overlaid track is *slaved* to that of the other tracks. What movie tracks bring to the table is the ability to have *non-slaved* time bases in a single movie. That is to say, each child movie can have its own time base, resulting in looping and playback rate characteristics independent of those of the parent movie.

Adding a Movie Track to a Movie

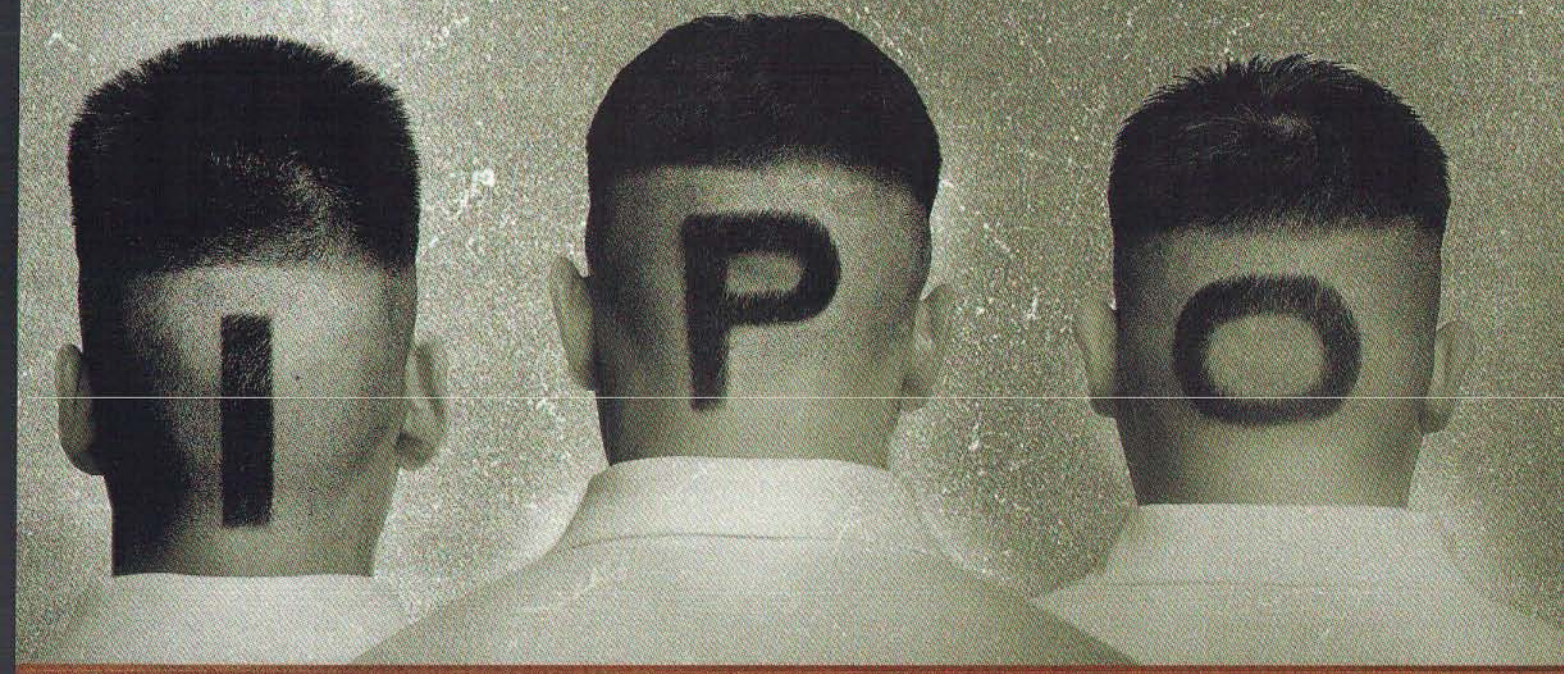
So let's see how to create movie tracks. Suppose that theWindowObject is a window object for an open QuickTime movie file and that theDataRef and theDataRefType are a data reference and a data reference type for some other QuickTime movie file. Then we can call the QTMIM_AddMovieTrack function defined in Listing 11 to add to that open movie a movie track that references that file. (Once again, we'll postpone discussing data references until the next article; for now, all we need to know is that they pick out QuickTime movie files, either on the local machine or elsewhere on the Internet.)

Listing 11: Adding a movie track to a QuickTime movie

QTMIM_AddMovieTrack

```
OSErr QTMIM_AddMovieTrack (WindowObject theWindowObject,
                  OSType theDataRefType, Handle theDataRef)
{
  Movie       myMovie = NULL;       // the parent movie
  Track       myTrack = NULL;       // the movie track
  Media       myMedia = NULL;       // the movie track's media
  OSErr       myErr = paramErr;
```

BOOTCAMP FOR STARTUPS.℠
THE FEW. THE PROUD. THE OBSESSED.



LONDON, September 4–5 **BOSTON, September 20–21**

garage.com
we start up startups

Attention. Join Garage.com's two-day Bootcamp for Startups. Learn the fundamentals of taking your company from startup to IPO. Hear from the high tech industry's top investors, experts, and entrepreneurs. Gain invaluable information about raising capital, building a buzz, hiring top talent, and launching your product. At ease.

LOG ON TO **WWW.GARAGE.COM/BOOTCAMP** TO LEARN MORE & REGISTER TODAY.

Forbes CAPITALIST TOOL
SAFEGUARD
TESTA, HURWITZ & THIBEAULT, LLP
hp invent
Goldman Sachs
TechSpaceXchange
ADVANCED TECHNOLOGY VENTURES
METRIUS
PRICEWATERHOUSECOOPERS
quidnunc
Silicon Valley Bank
plural
THESTANDARD
IBM

```
  if ((theWindowObject == NULL) ||
                              (theDataRef == NULL))
    goto bail;

  myMovie = (**theWindowObject).fMovie;

  // create the movie track and media
  myTrack = NewMovieTrack(myMovie,
                    FixRatio(kChildMovieWidth, 1),
                    FixRatio(kChildMovieHeight, 1),
                      kFullVolume);
  myErr = GetMoviesError();
  if (myErr != noErr)
    goto bail;

  myMedia = NewTrackMedia(myTrack, MovieMediaType,
                          kMovieTimeScale, NULL, 0);
  myErr = GetMoviesError();
  if (myErr != noErr)
    goto bail;

  // create the media sample(s)
  myErr = BeginMediaEdits(myMedia);
  if (myErr != noErr)
    goto bail;

  myErr =
QTMIM_AddMovieTrackSampleToMedia(theWindowObject,
              myMedia, theDataRefType, theDataRef);
  if (myErr != noErr)
    goto bail;

  myErr = EndMediaEdits(myMedia);
  if (myErr != noErr)
    goto bail;

  // add the media to the track
  myErr = InsertMediaIntoTrack(myTrack, 0, 0,
                GetMediaDuration(myMedia), fixed1);

bail:
  return(myErr);
}
```

There is absolutely nothing new about this function. It's virtually identical to the function QTMM_CreateVideoMovie that we encountered in an earlier article (see "Making Movies" in *MacTech*, June 2000). The only real difference is that we've created a media of type MovieMediaType; also, here we call QTMIM_AddMovieTrackSampleToMedia to add media samples to the new track, while earlier we called QTMM_AddVideoSamplesToMedia.

Creating a Movie Track Media Sample

By now you might be wondering what this has to do with atom containers. The answer is simple: the media sample for a movie track consists of an atom container whose atoms specify the movie to be embedded in the main movie, as well as some of the playback characteristics of the embedded movie. In other words, the function QTMIM_AddMovieTrackSampleToMedia needs only to create an appropriate atom container and pass that container to the AddMediaSample function.

We'll begin therefore by calling QTNewAtomContainer to create a new atom container; since this container will serve as our media sample, we'll call it mySample:

```
myErr = QTNewAtomContainer(&mySample);
```

Into this new atom container we want to put an atom of type kMovieMediaDataReference, whose data consists of the data reference type and the data reference of the movie file that is to be the embedded movie. We can create the atom data like this:

```
myData = NewPtrClear(sizeof(OSType) +
                      GetHandleSize(theDataRef));
myType = EndianU32_NtoB(theDataRefType);
BlockMove(&myType, myData, sizeof(OSType));
BlockMove(*theDataRef, myData + sizeof(OSType),
                       GetHandleSize(theDataRef));
```

Then we can insert the atom into the atom container by calling QTInsertChild:

```
myErr = QTInsertChild(mySample,
kParentAtomIsContainer,
                    kMovieMediaDataReference, 1, 1,
                  GetPtrSize(myData), myData, NULL);
```

At this point, we could call AddMediaSample to add the atom container mySample as the single media sample of the movie track. But we'd like the embedded movie to start playing automatically when the parent movie reaches the start time of the movie track, which is not the default behavior. To have the embedded movie automatically start playing, we need to add another atom to the atom container, of type kMovieMediaAutoPlay.

```
myBoolean = true;
myErr = QTInsertChild(mySample,
kParentAtomIsContainer,
                    kMovieMediaAutoPlay, 1, 1,
              sizeof(myBoolean), &myBoolean, NULL);
```

Now we can create a sample description and add the atom container to the movie track media. Listing 12 shows our function QTMIM_AddMovieTrackSampleToMedia for adding a media sample to a movie track.

Listing 12: Adding a sample to the movie track media

QTMIM_AddMovieTrackSampleToMedia

```
OSErr QTMIM_AddMovieTrackSampleToMedia
      (WindowObject theWindowObject, Media theMedia,
          OSType theDataRefType, Handle theDataRef)
{
#pragma unused(theWindowObject)
  QTAtomContainer             mySample = NULL;
  QTAtom                        myRegionAtom;
  SampleDescriptionHandle     myImageDesc = NULL;
  Ptr                           myData = NULL;
  OSType                        myType;
  Boolean                       myBoolean;
  OSErr                         myErr = paramErr;
```

```
  // create a new atom container to hold the sample data
  myErr = QTNewAtomContainer(&mySample);
  if (myErr != noErr)

    goto bail;

  // concatenate the data reference type and data reference
  // into a single block of data
  myData = NewPtrClear(sizeof(OSType) +
                       GetHandleSize(theDataRef));
  if (myData == NULL)

    goto bail;

  // convert the data to big-endian format
  myType = EndianU32_NtoB(theDataRefType);

  BlockMove(&myType, myData, sizeof(OSType));
  BlockMove(*theDataRef, myData + sizeof(OSType),
                       GetHandleSize(theDataRef));

  // add an atom of type kMovieMediaDataReference
  // to the atom container
  myErr = QTInsertChild(mySample,
                      kParentAtomIsContainer,

                  kMovieMediaDataReference, 1, 1,
                  GetPtrSize(myData), myData, NULL);
  if (myErr != noErr)
    goto bail;

  // add an auto-start atom

  myBoolean = true;
  myErr = QTInsertChild
                  (mySample, kParentAtomIsContainer,
              kMovieMediaAutoPlay, 1, 1,
             sizeof(myBoolean), &myBoolean, NULL);
  if (myErr != noErr)

    goto bail;

  // create a sample description
  myImageDesc = (SampleDescriptionHandle)
          NewHandleClear(sizeof(SampleDescription));
  if (myImageDesc == NULL)

    goto bail;

  (**myImageDesc).descSize =
sizeof(SampleDescription);
  (**myImageDesc).dataFormat = MovieMediaType;

  myErr = AddMediaSample(
          theMedia,
          mySample,
          0,
          GetHandleSize((Handle)mySample),
  GetMovieDuration(GetTrackMovie(GetMediaTrack
                (theMedia))),          myImageDesc,
          1,
          0,

          NULL);

bail:

  if (myData != NULL)
    DisposePtr(myData);

  if (myImageDesc != NULL)
    DisposeHandle((Handle)myImageDesc);

  if (mySample != NULL)
    QTDisposeAtomContainer(mySample);

  return(myErr);
}
```

Notice that, as promised earlier, we call GetHandleSize to get the size of the atom container when we call AddMediaSample.

This Month's Code

This month, our sample code is scattered across four different sets of files. The updated version of QTInfo is called QTInfoPlus and includes a new version of the QTInfo_MakeFilePreview function and all the utilities that we used to access the atoms in a QuickTime movie file. The code for creating shortcut movie files is contained in the snippet QTShortcut.c. The code for getting and setting the user's Internet connection speed preference is contained in the file QTUtilities.c (which has been part of every project we've developed so far). Finally, the code for adding movie tracks to QuickTime movies is contained in the project for the sample application QTMovieTrack. QTMovieTrack allows the user to configure a large number of settings for the new movie track, in addition to the auto-playback setting (which we considered earlier). **Figure 6** shows the dialog box that QTMovieTrack displays to allow the user to configure a new movie track.

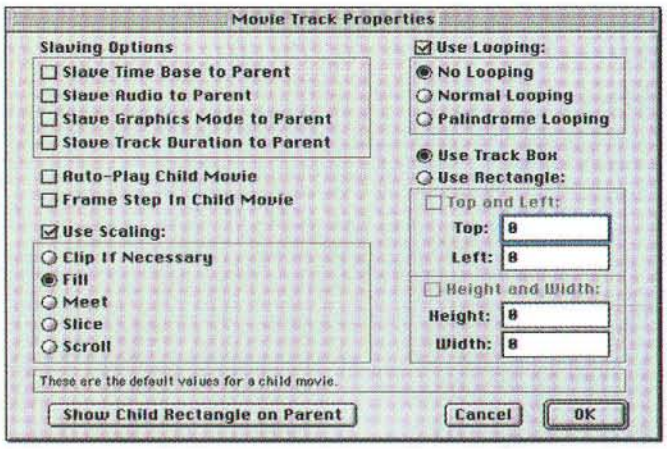


Figure 6. *QTMovieTrack's Movie Track Properties dialog box.*

For a complete explanation of the slaving and scaling options illustrated in **Figure 6**, see the document "QuickTime 4.1", downloadable in PDF form from the location <http://developer.apple.com/techpubs/quicktime/qtdevdocs/RM/rmWhatsnewQT.htm>.

Conclusion

QuickTime tries very hard to insulate us from having to work directly with the kinds of atoms that comprise movie files (the so-called "classic" or "chunk" atoms). The Movie Toolbox provides an extensive set of high-level routines that we can use to create new movie files, open existing movie files, edit movie files, and so forth. Nonetheless, we've seen that there are occasions when we do need to interact with atoms directly. A good example of this concerns adding a file preview to a single-fork movie file. QuickTime currently provides no API to do this, so we are forced to work with the file data — the atoms — directly. Similarly, if we want to create a shortcut movie file on a machine that's running a version of QuickTime prior to 4.0, we need to work with atoms.

By contrast, we'll encounter atom containers and their associated atom container atoms at virtually every step we take forward in our journey through QuickTime. Atom containers and their children provide an easy way to maintain hierarchical data, and they're backed by an extensive programming interface. So atom containers are now the repository of choice for storing and exchanging data. We'll see them used in media samples, tween tracks, input maps, musical instruments, wired actions, video effect tracks, and for a large number of other uses. In other words, Internet connection speed preferences and movie tracks are just the tip of the ice floe.

Credits

The functions for getting and setting a user's Internet connection speed preferences are based on code by Mike Dodd in the Letter from the Ice Floe, Dispatch 17 (found at <http://developer.apple.com/quicktime/icefloe/dispatch017.html>. Thanks are due to Ken Doyle for reviewing this article and offering some helpful comments. MT

We're giving you four days in October to catch up to the future.

Announcing QuickTime Live! October 9-12 at the Beverly Hilton Hotel. With over 50 million downloads, QuickTime™ is fast becoming the industry standard for multimedia content on the Mac® OS and Windows. At QuickTime Live! we're presenting four days of exhibits, workshops and conferences to help you put QuickTime to work brilliantly–from CD content to web streaming. If you're a multimedia pro, it's your future. Don't miss it. For details and registration information, visit our website, www.apple.com/quicktimelive.

By Erik Sea

Speaking to your Software

Making your application work well with IBM ViaVoice Enhanced Edition 2.0

What Can I Say?

It's here! Talking to machines and having them respond and react has been the stuff of science fiction for decades. The promise has been so long in coming that the release of ViaVoice Millennium for Mac last year seemed to take some people by surprise — many a passerby at MacWorld San Francisco was astonished by the speed and accuracy of the system, even in noisy showfloor conditions. Nonetheless, the combination of computational power and algorithm design has finally produced speech recognition software for the Mac that permit routine and productive use, especially as fast, new copper IBM PowerPC chips find their way into more and more Macs.

ViaVoice Millennium, the first release, was a low-end product, providing dictation into a single application, SpeakPad, and non-customizable transfer scripts. Good for basic dictation, with a large, extensible vocabulary, dictation macros, and AppleScript support. ViaVoice Enhanced builds on this capability, adding new features such as direct dictation into selected applications and allowing customization of "built-in" functions through AppleScript.

"Aha!" you say — "Direct dictation into selected applications, but what if I'm not among the 'selected' few?" Fair enough — IBM can only test and support a few high-profile programs (although the development team is always interested in testing new software for compatibility, particularly games). However, the ViaVoice software doesn't *prevent* dictation into any application and, in many cases, the Mac OS and ViaVoice extensions that ship with our software are all you need — *your application may already support dictation and correction without you writing a single line of code!*

Probably, though, you should write a line or two of code. This is essential for maintaining the awe and admiration from your employer, and I know that you really do want to anyway.

ViaVoice Speech Technology

But, before we write code, let's talk about speech. Or speak about talk, and how the ViaVoice engine decides what words it thinks you uttered.

Unlike earlier "discrete" speech recognition systems, which ...required ... distinct ... pauses ... between... words, ViaVoice works with "continuous" speech, with no unnatural breaks between words. In consumer products, we're not quite to the stage where you can have conversations with your computer, or even record or transcribe a speech or a meeting, but for one person, sitting at a computer, speaking clearly and providing cues such as punctuation and formatting, recognition is really quite good. In any case, there are other technologies that will need some work before you can say, "Tea, Earl Grey, Hot" and get what you would like.

Training

Recognition accuracy is also improved by training, which allows ViaVoice to construct a mathematical picture of your voice, which it can then use with its models.

The user reads a prepared story is read to the system to train it — the system knows what the words are, and what they ought to sound like. By comparing these sounds to actual sounds, a difference can be calculated down to the individual sounds that make up a

Erik has been working on Mac development throughout modern history, and, in that time, has done everything from drivers to GUI, and from telecom to graphics processing. Last year, in search of new challenges, he joined the ViaVoice for Mac team at IBM in Florida, and has led the recent release of the Enhanced Edition, which, it is not commonly known, is written specifically for the Mac from the ground up. In his spare time, he breeds noncarnivorous slinkys in the desks of unsuspecting coworkers. You can reach Erik at esea@us.ibm.com.

word. For example, suppose when I talk I make my Ts sound like Ds much of the time: "butter" sounds more like "budder" when I say it, but someone else my very crisply say "butter".

Once this picture of my voice exists, ViaVoice can predict with some level of accuracy how I might say a specific word.

Vocabularies and Language Models

The ViaVoice vocabularies (sometimes called "dictionaries") and Language Models are basically large databases of word pronunciations and word positions relative to other words, respectively. You can add your own words to the vocabulary, and teach ViaVoice what they sound like — an extension to the spell dictionary sort of operation you may be familiar with for wordprocessors.

Language models are a bit trickier to explain. Start with a large number of typical dictation documents, feed them into a shredder, and out pops a language model at the other end. Well, maybe not a shredder. The secret is they are dissected by elves. Honest.

However they are created, you can think of a language model as collections of trigram (comprising three consecutive words). The probability that a given spoken word really is a specific written word is influenced by the word around it, and trigrams capture this relationship. By no means should you equate "language model" with "grammar" — they are not at all the same, as grammars reflect complex usage rules that are difficult even for most humans to apply correctly all the time.

To illustrate further, consider the sentence "Please write to Mr. Wright right away." We hand this to the elves, who produce trigrams along the lines of those shown in Figure 1. Now, assuming that you don't pronounce "write", "Wright", and "right" in noticeably different ways, how does this system figure out which sounds correspond to which words? The trigrams from this sentence, combined with other trigrams from other sentences, end up with results like "If it's preceded by 'Mr.', it's most likely 'Wright'", and so on. ViaVoice is very good at this kind of thing and, where it makes mistakes, it can even learn not to make them in the future through correction, which improves accuracy.

Sample Sentence:	Please write to Mr. Wright right away.
Resulting Trigrams:	<Please,write,to> <write,to,Mr.> <to,Mr.,Wright> <Mr.,Wright,right> <Wright,right,away> <right,away,[period]>

Figure 1. *Basic Trigram Construction.*

By the way, you might want to take a moment to consider the word "to" in the Figure 1 example. How does ViaVoice know the difference between "to" and "two" and "too" when they all sound the same? The answer, once again, is by context, as captured in the trigrams!

The more you use it, the better *it* gets

"I find it interesting that the software is learning about me while I am learning about it."

This customer remark is based on the realization that, as you use ViaVoice, it actually continues to improve your voice model. For example, if you add words that the system doesn't know, those get added to the model. If you make corrections using the correction window **(See Figure 2)**, those corrections get applied to the voice model.

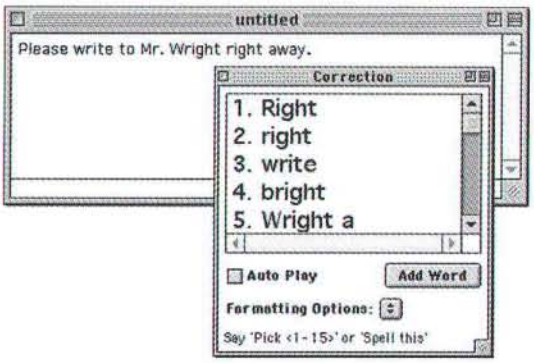


Figure 2. *Correction Window — if your associate is actually Mr. "Right".*

Also, if you have text documents containing phrases, words, and names that you expect to dictate often, you can analyze those documents, which will further update the voice models. ViaVoice Enhanced has extra facilities that integrate the document analysis features into dictation, so that you don't need to store up documents and run the analysis program yourself.

It is not unusual to hear of accuracy improving from 95% to 98% with persistent use.

Speaking of Japanese

And now, on to the code, and how ViaVoice adopts and extends the Mac OS to bring you dictation as seamlessly as Japanese.

The Mac was probably the first platform to make internationalization a key design objective. As a result, many components of Mac OS, including almost all toolbox functions, are ready-made compatible with other languages and script systems.

You've probably all dealt with localization issues before, ranging from not hardcoding strings to not making assumptions about the size of the label on a button when translated. Some of you have no doubt dealt with the multibyte matters arising from making a product work correctly with Japanese, ranging from not being able to make byte = character assumptions to working nicely with Input Methods.

With the distribution of Mac OS 9, it became easier for any Mac user to install multiple input methods, for languages such as Japanese or Korean or different eastern European script systems as well as the "default" (typically Roman) keyboard system. Previously, while such capabilities were available, input methods were not widely used outside of countries where they were absolutely required.

An input method traditionally allows you to enter text in a different script system by typing a few characters, pulling up palettes based on those characters, and selecting similar symbols from those palettes.

There are two forms of input associated with an input method: inline and bottomline, shown in Figures 3 and 4 respectively. Inline input is generally preferred by users; bottomline input requires you to enter data in one place and have it show up in another place, in another font. Bottomline input also lacks the ability to go back and edit later.

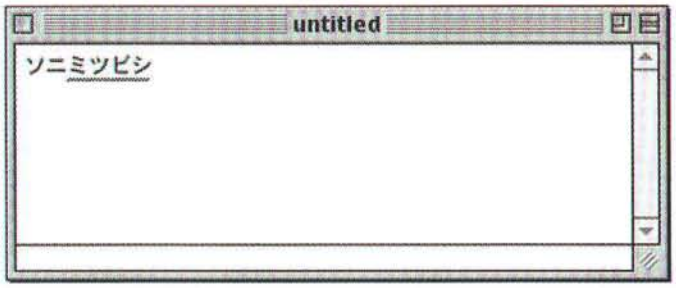


***Figure 3**. Direct inline support in Japanese.*

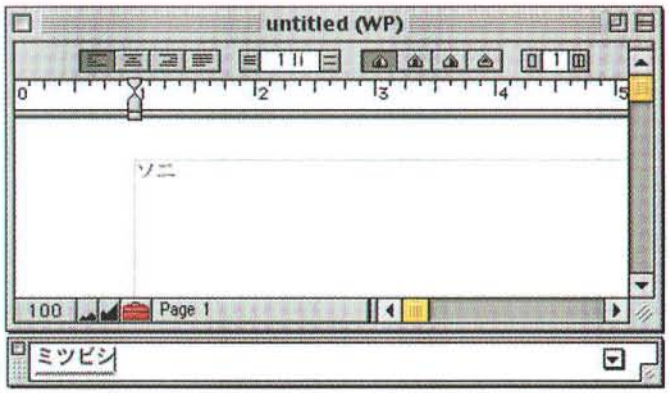


***Figure 4**. Japanese bottomline input (text entered in lower window).*

In designing ViaVoice Enhanced direct dictation, we decided that we could use the input method architecture developed by Apple for non-Roman languages like Japanese, and use it for word-based input and correction. We did end up needing to extend the model slightly, as I'll describe later, but we did it in the background so that, in some circumstances, if you do the work to allow Japanese inline input, you also get ViaVoice speech inline input and correction for free!

TSM, TSMTE, IBM VV & U

These acronyms represent the key players in IBM ViaVoice dictation into any application.

Introduced way back in Mac OS 7.1, the Text Services Manager (TSM) has provided functionality for other languages (primarily multibyte languages) for years. With TSM, a savvy developer could write a few lines of initialization code, and then install 4 Apple event handlers that, much like AppleScript, performed operations like inserting text, showing and hiding the bottomline input window, and telling the input method where a given text offset was. TSM is well-documented in *Inside Macintosh: Text*, forming all of chapter 8.

Later, Apple introduced the Text Services Manager for TextEdit (TSMTE), which eliminated the need to write any of the event handling code in applications that only used TextEdit — from this point, it was only necessary to initialize the manager and let TSMTE handle the rest. This functionality is well documented in Apple's *Tech Note TE27*.

Full inline input was not achieved until a new fifth Apple event was added. This is the GetText or 'gtxt' event, and it is only documented in a *develop* article by Tague Griffith (issue 29), or in Apple's *Tech Note 10002* (which is only available in Japanese).

For speech, we determined that the above was not enough. While we could have gone our own way with a completely different model and then tried to sell it to developers, we decided to, instead, augment the existing TSM calls, by adding a couple of extra parameters to GetText, and adding another event which we call SetSelection. With just these two changes, we have the necessary and sufficient conditions for dictation and correction. Sure, we could do more with more events (and, may extend the system to enhance functionality in the future), but you're busy trying to figure out how to get your software to run under Carbon, so we thought we'd cut you a break! Oh, and as you may have inferred, if you've relied on TSMTE for Inline Support in Japanese, the changes to GetText and the addition of SetSelection is done for you automatically. We'll talk about these additions later when I present the implementation code.

If you don't use TextEdit exclusively, you cannot rely on TSMTE for Japanese input, and you will likewise need to do the work of handling the calls and adding the parameters yourself. But, even so, if you've done the work for Japanese (Japan being the second largest Mac market in the world), the incremental work for adding ViaVoice support is bordering on the trivial! *As I write this, it may also be necessary to write these handlers for Mac OS X, whether or not you use TextEdit. By the time you read this, we'll have a better idea of what the story is. Either way, the solution is not difficult, it's just a matter of not knowing which path will be required for Mac OS X.*

Adding the Ears

By now, you're probably frothing at the prospect of dictation-enabling your code. Let's get right to it. In Listing 1, you'll see how to enable TSM as part of your startup retinue and disable it as part of quitting (Carbon Applications don't need to do this — the OS does it for you automatically). While you're enabling TSM, know that while you must set the "high level event aware" bit in the SIZE resource, you do not need to set the bit "Use text edit services", because it is deprecated (relates actually to an earlier implementation prior to TSMTE).

Listing 1: Becoming TSM Aware

Determining if TSM is available, initializing it, and cleaning up

This is more or less boilerplate code that is required for any application. Note that, under Carbon, you do not even need to make these calls, as the system will do them for you — that is, any Carbon application is TSM-aware without any special calls!

```
Boolean IsTSMAvailable (void) {

   SInt32      version;
   Boolean     available= false;

   // Note: gestaltTSMgrAttr is not defined under Mac OS 9
   // so we use the gestaltTSMgrVersion selector instead...

   if (noErr == Gestalt (gestaltTSMgrVersion, &version)) {
     if (version >= gestaltTSMgr15) {
       available = true;
     } // if
   } // if

   return available;

} // IsTSMAvailable

void StartTSM (void) {

   // Initialize TSM, and install our event handlers...

   OSErr     err     = noErr;

   if (IsTSMAvailable ()) {

     #if TARGET_API_MAC_OS8
       err = InitTSMAwareApplication ();
     #endif

     // Install TSM event handlers here — see later section

   } // if

   gFontForce = GetScriptManagerVariable (smFontForce);
   SetScriptManagerVariable (smFontForce, false);

} // StartTSM

void CloseTSM (void) {

   // Clean up all TSM things, including our event handlers...

   SetScriptManagerVariable (smFontForce, gFontForce);

   if (IsTSMAvailable ()) {

     #if TARGET_API_MAC_OS8
       (void) CloseTSMAwareApplication ();
     #endif

     // Remove AE handlers here...

   } // if

} // CloseTSM
```

You'll notice that there is no special "is ViaVoice installed" code. Again, because we're an input method, the code you write works whether we're installed or not!

If you're only using TextEdit and dialog boxes, set the refcon field of your dialogs to kTSMTEInterfaceType and you're done. Go talk to your computer for a while. Tell your friends/family/coworkers I said it was OK.

Beyond TextEdit Support

Although many applications can live with just TextEdit, the 32K limit, among other things, lead many people to roll their

own or use a third-party code library such as WASTE (although now Apple has made a special MLTE — multi-lingual text edit — available). These will require implementation of five Apple event handlers. The complexity of these handlers, naturally, depends on how your code is laid out, but in general, you're just providing an external API to functions or data that you already have written. And you need to do 98% of this for Japanese anyway, so why not squeeze in the 2% for speech? I'll even write the code for you. Alternatively, you can use WASTE (WorldScript-Aware Styled Text Engine, by Marco Piovanelli) which is available in source code form all over the Web, and does much of the work (WASTE would need to be adjusted slightly to handle some of the modified events described here).

But before we delve into the handlers themselves, some TSM terminology. The basics of TSM are discussed in chapter 7 of *Inside Macintosh: Text*, and also in Tague Griffith's article in *develop* 29 (see links section), so I will gloss over most of that — the code is pretty self explanatory when read with those references in hand. Ironically, Tague even suggests that input methods might one day be used for dictation input!

TSM keeps track of things on a document basis, where a document is a unique editable area of text. You'll need to add some extra handling to your event loop, so that TSM gets a crack at events it may need to intercept with TSMEvent(), and give TSM first crack at menu events with TSMMenuSelect() (you may have noticed that input methods typically have menus — the ViaVoice input method does not have a menu, but you do want to support Japanese too, right?). As well, when your TSM-aware documents become active and inactive, you need to tell TSM with ActivateTSMDocument() and DeactivateTSMDocument().

Input methods also might like to change the cursor (currently, the ViaVoice input method does not do so, but others do and ViaVoice may in the future), so in your cursor-management routines, or at idle time, call SetTSMCursor(). For this to work, your mouse-moved region (the final parameter to WaitNextEvent() which most people lazily set to NULL) needs to be a single point — since you have no way of knowing when TSM wants to change the cursor.

OK, that was pretty fast, but as it's been written before in the references above, I didn't want to repeat it. You can look at the sample application that comes with this article if you're lost.

The key part to dictation-enablement is the Apple event handlers. These handlers need to get installed for the input methods, including ViaVoice, to be able to extract information from your document content. ViaVoice doesn't vary much from the standard architecture, and I will highlight the differences.

Position to Offset Event

This event converts screen coordinates into an offset in your document. You receive a point, and return an offset. ViaVoice does not currently use this event.

```
OSErr
DoPos2Offset (DialogPtr inDialog, const AppleEvent*
              inAppleEvent, AppleEvent* outReply) {

  Size          actualSize;
  DescType      actualType;
  OSErr         err        = noErr;
  Boolean       dragging   = false;
  Point         currentPoint;
  SInt32        offset;
  SInt16        where;
  DialogItemType dialogType;
  Handle        dialogHandle;
  Rect          dialogBounds;
  GrafPtr       svPort;

  GetPort (&svPort);
  #if TARGET_API_MAC_OS8
    SetPort (inDialog);
  #else
    SetPortDialogPort (inDialog);
  #endif

  // Required parameter is a point...

  if (err == noErr) {
    err = AEGetParamPtr (inAppleEvent, keyAECurrentPoint,
            typeQDPoint, &actualType, &currentPoint,
            sizeof (currentPoint), &actualSize);
  } // if

  // Optional parameter is for dragging...

  if (err == noErr) {
    (void) AEGetParamPtr (inAppleEvent, keyAEDragging,
              typeBoolean, &actualType, &dragging,
              sizeof (dragging), &actualSize);
  } // if

  // Now, we should do all sorts of calculations, but,
  // TextEdit will more or less do this for us once we
  // figure out if it's in the right place...

  GlobalToLocal (&currentPoint);
  GetDialogItem (inDialog, kEditTextDialogItem,
                 &dialogType, &dialogHandle, &dialogBounds);
  if (PtInRect (currentPoint, &dialogBounds)) {
    TEHandle    dialogTE = GetDialogTEHandle (inDialog);
    offset = TEGetOffset (currentPoint, dialogTE);
    where = kTSMInsideOfActiveInputArea;
  } else {
    where = kTSMInsideOfActiveInputArea;
  } // if

  // Stuff the return values here

  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAEOffset,
          typeLongInteger, &offset, sizeof (offset));
  } // if
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAERegionClass,
          typeShortInteger, &where, sizeof (where));
  } // if

  SetPort (svPort);

  return err;

} // DoPos2Offset
```

Offset to Position Event

The reverse of Position to Offset: return a global point given a text offset. ViaVoice does not currently use this event.

```
OSErr
DoOffset2Pos (DialogPtr inDialog, const AppleEvent*
      inAppleEvent, AppleEvent* outReply) {

  Size          actualSize;
  DescType      actualType;
  OSErr         err        = noErr;
  SInt32        offset;
```

```
  GrafPtr         svPort;
  Point           thePoint;
  TEHandle        teHandle    = GetDialogTEHandle (inDialog);
  Rect            bounds;

  GetPort (&svPort);
  #if TARGET_API_MAC_OS8
    SetPort (inDialog);
    bounds = inDialog->portRect;
  #else
    SetPortDialogPort (inDialog);
    GetPortBounds (GetDialogPort (inDialog), &bounds);
  #endif

  // Required parameter is an offset position...

  if (err == noErr) {
    err = AEGetParamPtr (inAppleEvent, keyAEOffset,
            typeLongInteger, &actualType, &offset,
            sizeof (offset), &actualSize);
  } // if

  // Convert the offset to a position, taking into
  // account whether it's visible or not...

  if (err == noErr) {
    thePoint = TEGetPoint (offset, teHandle);
    if ((offset < 0) && (offset > (**teHandle).teLength)) {
      err = errOffsetInvalid;
    } else if (PtInRect (thePoint, &bounds)) {
      err = errOffsetIsOutsideOfView;
    } // if
  } // if

  // Return the point (in global coordinates), and
  // the parameters of the text...

  if (err == noErr) {
    LocalToGlobal (&thePoint);
  } // if

  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAEPoint, typeQDPoint,
          &thePoint, sizeof (thePoint));
  } // if
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAETextFont,
            typeLongInteger, &(**teHandle).txFont,
            sizeof (SInt32));
  } // if
  if (err == noErr) {
    Fixed theFixed = Long2Fix((**teHandle).txSize);
    err = AEPutParamPtr (outReply, keyAETextPointSize,
            typeFixed, &theFixed, sizeof (theFixed));
  } // if
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAETextLineHeight,
            typeShortInteger, &(**teHandle).lineHeight,
            sizeof (SInt16));
  } // if
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAETextLineAscent,
            typeShortInteger, &(**teHandle).fontAscent,
            sizeof (SInt16));
  } // if

  SetPort (svPort);

  return err;

} // DoOffset2Pos
```

Update Active Input Area Event

This event is used to hilite an area of your document as requested by the input method. ViaVoice does not currently use this event.

```
OSErr
DoUpdateActiveInputArea (DialogPtr inDialog, const
      AppleEvent* inAppleEvent, AppleEvent* /*outReply*/) {

  Size          actualSize;
  DescType      actualType;
  OSErr         err           = noErr;
  AEDesc        theTextDesc   = {};
  AEDesc        theHiliteDesc = {};
  AEDesc        theUpdateDesc = {};
  SInt32        fixLength;
  TextRange     thePinRange;
  TEHandle      teHandle      = GetDialogTEHandle (inDialog);
  ScriptLanguageRecord scriptCode;

  // Required parameters containing firmed text, script,
  // and fixed length...

  if (err == noErr) {
    err = AEGetParamDesc (inAppleEvent, keyAETheData,
            typeChar, &theTextDesc);
  } // if
  if (err == noErr) {
    // Note: "Inside Macintosh - Text" says this parameter
    // is under keyAEScriptTag, but in practice it appears to
    // be under keyAETSMScriptTag...
    err = AEGetParamPtr (inAppleEvent, keyAETSMScriptTag,
            typeIntlWritingCode, &actualType, &scriptCode,
            sizeof (scriptCode), &actualSize);
  } // if
  if (err == noErr) {
    // Note: "Inside Macintosh - Text" says this parameter
    // is required, but in reality, it seems to be optional
    // and not sent (and redundant with the actual size of
    // the data in theTextDesc) — we won't use or rely
    // on it...
    (void) AEGetParamPtr (inAppleEvent, keyAEFixLength,
              typeLongInteger, &actualType, &fixLength,
              sizeof (fixLength), &actualSize);
  } // if

  // Optional parameters hilite range list, update range,
  // and Pin range; we don't use any of these...

  if (err == noErr) {
    (void) AEGetParamDesc (inAppleEvent, keyAEHiliteRange,
             typeTextRangeArray, &theHiliteDesc);
  } // if
  if (err == noErr) {
    (void) AEGetParamDesc (inAppleEvent, keyAEUpdateRange,
             typeTextRangeArray, &theUpdateDesc);
  } // if
  if (err == noErr) {
    (void) AEGetParamPtr (inAppleEvent, keyAEPinRange,
             typeTextRange, &actualType, &thePinRange,
             sizeof (thePinRange), &actualSize);
  } // if

  // At this point, we need to be inserting text,
  // most probably...

  if (err == noErr) {
    #if TARGET_API_MAC_OS8
      SInt8     hState;
      Hstate = HGetState ((Handle) theTextDesc.dataHandle);
      HLock ((Handle) theTextDesc.dataHandle);
      TEDelete (teHandle); // Clean first...
      TEInsert (*(theTextDesc.dataHandle), GetHandleSize
        ((Handle) theTextDesc.dataHandle), teHandle);
      HSetState ((Handle) theTextDesc.dataHandle, hState);
    #else
      // AEDescs are opaque under Carbon. So we need
      // to allocate and copy using the accessor APIs.
      // OK, fine...
      Size      dataSize = AEGetDescDataSize (&theTextDesc);
      Handle    dataCopy = NewHandle (dataSize);
      if (dataCopy != NULL) {
        HLock (dataCopy);
        err = AEGetDescData (&theTextDesc, *dataCopy,
                dataSize);
      } else {
        err = memFullErr;
      } // if
      if (err == noErr) {
        TEDelete (teHandle); // Clean first...
        TEInsert (*dataCopy, dataSize, teHandle);
      } // if
```

```
    if (dataCopy != NULL) {
      DisposeHandle (dataCopy);
    } // if
  #endif
} // if

// Clean up...

(void) AEDisposeDesc (&theTextDesc);
(void) AEDisposeDesc (&theHiliteDesc);
(void) AEDisposeDesc (&theUpdateDesc);

return err;

} // DoUpdateActiveInputArea
```

Get Text Event

The GetText event is a mystical event introduced by Apple Japan and, until recently, documented only in Japanese. This event allows the input method to request the application to return text that has already been committed to the document. ViaVoice uses this event to extract text for correction.

More than that, however, ViaVoice expects two additional parameters: the offset start and the offset end. Why is this? Because, unlike simple text-editing Input Methods, ViaVoice distinguishes between the first utterance of the word "the" and the second — it actually keeps track of all the dictated text for a session, the relative words, and so on, in order to make correction work. If ViaVoice were to allow correction simply based on the text of the word, all of the additional contextual information and even the audio data would be useless!

Thankfully, if you use TextEdit, the extra parameters are added for you by ViaVoice, but for other wordprocessing situations, you'll need to add them. Relatively painless, in most cases, since you probably know the offsets of the selection anyway!

```
OSErr
DoGetSelectedText (DialogPtr inDialog, const AppleEvent*
      /*inAppleEvent*/, AppleEvent* outReply) {

  OSErr         err         = noErr;
  TEHandle      teHandle    = GetDialogTEHandle (inDialog);
  SInt8         hState      = HGetState ((Handle) teHandle);
  SInt64        selStart    = (**teHandle).selStart;
  SInt64        selEnd      = (**teHandle).selEnd;

  // The only required return is the current selected text...

  HLock ((Handle) teHandle);
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyAETheData, typeText,
          &(*teHandle)[selStart], selEnd-selStart);
  } // if

  // For ViaVoice, we also add the numeric values of the
  // start and end of the selection within the text...
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyVVStartSelectionParam,
          typeSInt64, &selStart, sizeof (selStart));
  } // if
  if (err == noErr) {
    err = AEPutParamPtr (outReply, keyVVEndSelectionParam,
          typeSInt64, &selEnd, sizeof (selEnd));
  } // if

  HSetState ((Handle) teHandle, hState);

  return err;
} // DoGetSelectedText
```

Set Selection Event

This event is new and unique to ViaVoice. For TextEdit, it is implemented for you in one of the ViaVoice extensions (with one caveat of course: if you don't expect the selection to change within your TextEdit fields, you may be surprised to see it change, or if you duplicate the selection range in one of your own data structures, you may end up out of sync).

Really, all this does is ask you application to change the active selection. This is necessary so that commands such as "correct 'the'" will work as the user expects them.

```
OSErr
DoSetSelection (DialogPtr inDialog, const AppleEvent*
      inAppleEvent, AppleEvent* /*outReply*/) {

  Size          actualSize;
  DescType      actualType;
  OSErr         err         = noErr;
  TEHandle      teHandle    = GetDialogTEHandle (inDialog);
  SInt64        selStart;
  SInt64        selEnd;
  Boolean       doDraw      = false;
  // This is a ViaVoice-specific event. Retrieve the
  // selection, and the optional draw event, and do it...
  if (err == noErr) {
    err = AEGetParamPtr (inAppleEvent,
          keyVVStartSelectionParam, typeSInt64, &actualType,
          &selStart, sizeof (selStart), &actualSize);
  } // if
  if (err == noErr) {
    err = AEGetParamPtr (inAppleEvent,
          keyVVEndSelectionParam, typeSInt64, &actualType,
          &selEnd, sizeof (selEnd), &actualSize);
  } // if
  if (err == noErr) {
    (void) AEGetParamPtr (inAppleEvent,
          keyVVDrawSelectionParam, typeBoolean, &actualType,
          &doDraw, sizeof (doDraw), &actualSize);
  } // if

  // Clip off the ends to TextEdit range...

  if (err == noErr) {
    if (selEnd > 0x7fff) {
      selEnd = 0x7fff;
    } // if
    if (selStart > 0x7fff) {
      selStart = 0x7fff;
    } // if
    if (selStart < 0) {
      selStart = 0;
      err = paramErr;
    } // if
    if (selEnd < 0) {
      selEnd = 0;
      err = paramErr;
    } // if
  } // if
  if (err == noErr) {
    TESetSelect (selStart & 0x7fff, selEnd & 0x7fff,
                 teHandle);
  } // if
  if ((err == noErr) && doDraw) {
    // We would optionally draw here, but TESetSelect does
    // that anyway, there's no point. The idea is that the
    // screen will flicker if you honor this optional
    // parameter...
  } // if

  return err;

} // DoSetSelection
```

Fine Tuning

The Mac OS is a cooperative multitasking system. ViaVoice direct dictation involves the cooperation of at least four different

applications, all of which need a slice of time. See **Figure 5** for an overview of how the components interact. The recognition engine, in fact, will shut off if it doesn't get enough time to handle the incoming audio stream which, like the real world, isn't very cooperative. Audio data is big, so the amount of time before a shutdown is small. You can simulate this by clicking on a menu in Mac OS while the microphone is on.

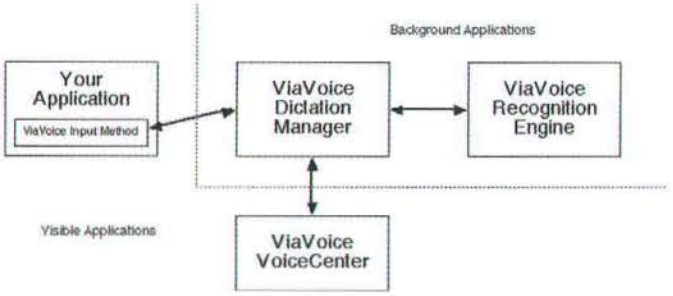


Figure 5. *Interprocess Communication within ViaVoice — and to your application — requires that everybody share the processor equitably!*

So, what you need to keep in mind that your application, when in the foreground, should be as friendly as it can be with the other processes, particularly calling WaitNextEvent frequently enough that the recognition engine gets time to process audio, the dictation manager gets enough time to assemble words and send them to your application, and the VoiceCenter gets enough time to communicate status and feedback to the user. Some applications try to avoid WaitNextEvent in order to improve their own apparent performance, but if you do this with ViaVoice, you won't get very good throughput, and you may even starve the engine to shutdown.

Not everything that you can type into is appropriate for dictation — sure, you could say that text is text, but dictation isn't really the same as data entry. Right now, there is no way to restrict dictation to numbers, or constrain the dictation search to a single word answer or a set of words that might be appropriate for a given field. Rather, right now, we're focussing on freeing up the keyboard and mouse so that the user can speak and think for prolonged periods in large bodies of text, like letters, email, or other prose. This is not for typing in a choice of eleven point text!

Test Drive

Bring up your application, start ViaVoice direct dictation services, activate the dictation system with the phrase "begin direct dictation", and then, when the system is ready, click in a text field of your application, and dictate the phrase "Please write to Mr. Wright right away [period]". After a couple seconds, you should see the text appear. If there are no errors, say "correct mister", and "Mr." will hilite, and the correction window will open with alternatives, as in **Figure 6**.

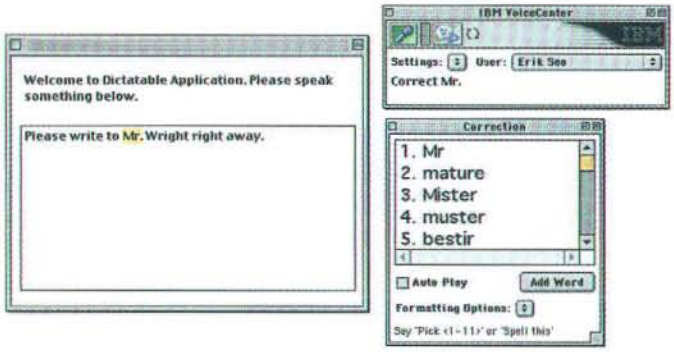


Figure 6. *Correcting in your application.*

Then, you can pick one of the alternatives, or say "close correction window". For more things to try, consult the ViaVoice Users Guide.

Getting Creative

Beyond direct dictation, there are other things you can do. You can write AppleScripts to control your application to perform routine operations. You could even have a "secret about box" phrase bring up a nice little Easter egg in your product. Likewise, you could have other key phrases that, rather than processing as text, trigger behaviors or commands. I'm sure there are other things that I've not thought of yet.

Future Directions

"Prediction is difficult, especially about the future."

A word about future versions. Simple extrapolation from ViaVoice Millennium late last year to Enhanced in the middle of this year should suggest that the ViaVoice for Mac team has been busy, and continues to be busy, adding features, fixing bugs, and getting the product into our customers' hands. I cannot say what will result from this activity, but it is likely that developer opportunities, already greatly expanded with Enhanced, will continue to grow as the product line itself evolves and matures.

An interesting point on this topic is that ViaVoice is the only speech software technology that is currently available for and has an installed base on Windows, Mac OS, and several flavors of Linux. If you're thinking cross-platform speech, this is where you want to be.

MicroPhone Off

There you have it. The amount of work you need to get dictation into your application varies from "it works already for free" to "I had to write a couple of Apple event handlers." Beyond that, you can get as creative as you want.

ViaVoice for Macintosh has been a best seller since its introduction in 1999, and with ubiquitous dictation availability in the latest edition, there's a good chance that many of your customers will have ViaVoice, and want to dictate into your application. Believe me, the number of comments about "I'd like to be able to dictate into Application X" exceeds just about any other feature request. Make it so!

Acknowledgements

I would like to thank **Deborah Grits**, **Eddie Epstein**, **Jeff Kusnitz**, and **Paul McLellan** for taking the time to give me feedback on this article as it was being written. My special thanks to the rest of the **ViaVoice for Mac team** who broke new ground on the Mac — twice — and helped bring the future closer.

Links

- http://www.ibm.com/viavoice/
  The main ViaVoice Web site.
- http://developer.apple.com/technotes/te/te_27.html
  Tech note TE 27 describing TSMTE functionality.
- http://developer.apple.com/dev/techsupport/develop/issue29/griffith.html
  Great article by Tague Griffith describing TSM and TSMTE (including the elusive 'gtxt' event).
- http://developer.apple.com/ja/technotes/tn10002.html
  Tech note describing 'gtxt' event. Only if you can read Japanese!
- http://developer.apple.com/techpubs/mac/Text/Text-409.html
  Chapter 7 of Inside Macintosh: Text - the essential reference for TSM functions and use.
- http://www.zdnet.com/pcmag/stories/reviews/0,6755,2385302,00.html
  PC Magazine review of ViaVoice showing test results of accuracy improvement to 98%. MT

Tonight's regularly scheduled programming will be preempted for a special episode of whatever the hell you want.

You won't believe what you can do with ReplayTV. It's not a VCR, it's a digital television recorder, so you can actually pause live television, and do your own live instant replays. It also has a search engine, so you can punch in a keyword, say "Golf," and it will find and record any golf program that comes on — all without videotape. Or you can just punch in the name of your favorite show and let ReplayTV find it and store every episode, so you'll never miss it again. If you had ReplayTV, what would you do? Call us at 877-replaytv or visit www.replaytv.com

replaytv™ some televisions have all the fun.

Available at any Tweeter Company, Best Buy and Amazon.com

By Andrew C Stone

More or Less: Drawers and Disclosure Views for Cocoa

In designing an intuitive interface for users, the great French architect Le Corbusier's adage still rings true: **less is more**. The less user interface you have visible, the more likely the user will be able to understand your program's feature set. But applications without depth and functionality are boring and leave the user feeling "if only the application could do X or Y". There are several ways to hide complexity and still have the features available for expert users. Along with the ubiquitous tabview, two major techniques are the drawer and the disclosure view.

Mac OS X introduces the concept of the drawer - a Dalicsque subwindow which expands from under the parent window in a smooth opening animation, remaining attached to the window until no longer needed when it animates smoothly closed. An example of a good use of drawers is Mail.app's "Mailboxes". Drawers have certain limitations however — for instance, you cannot pull out a bottom-mounted drawer with a depth greater than the height of the window since this would violate the physical reality being emulated. Moreover, there were no drawers in Mac OS 9 or Mac OS X Server, so another technique, disclosure views must be used. The disclosure view, the little blue triangle that "shows more" by expanding the window to reveal more user interface, is one excellent technique. You can see this button in the standard Save Panel - which expands to show the file browser or collapses to present a very simple save interface. In this article, you will learn how to implement drawers and two types of disclosure views: one that expands the window to the right, and one that expands the window below.

<<ClosedDrawer.tiff This window presents a simplified interface for the user, with the drawer closed.>> <<ExpandedDrawer.tiff When the drawer is pulled out, additional options become available>>

Drawing Upon Drawers

The drawer is an instance of the NSDrawer object, a simple non-visual controller type object that acts as a coordinator. NSDrawer has a simple application programmer's interface (API) which allows you to specify the parent window, the content or container view which gets inserted into the drawer, the preferred edge from which the drawer expands, methods to programmatically open and close it, and a method to determine whether the drawer is open, closed, opening or closing. Moreover, there are optional delegate methods sent to the drawer's delegate and objects which register for drawer notifications before and after closing and opening and before resizing of the drawer. The full API is presented in /System/Library/Frameworks/AppKit.framework/Headers/NSDrawer.h.

InterfaceBuilder - the application to build graphical user interfaces via drag and drop of components - now has two ways to create drawers without writing a single line of code. From IB's Windows Palette, you can either drag out an NSDrawer object, and hook up the parent and content view yourself, or drag out a window with the drawer window already attached. The first method is excellent if you are modifying an existing interface, and the latter when you are designing from scratch.<<DrawersPalette.tiff: You can create completely functional drawers just by drag and drop from InterfaceBuilder>>.

I recommend that you create a new IB document and add a drawer/window combination to see how easy it is, and to learn a trick of the trade: the invisible box grouping technique. All visible objects in Cocoa are NSView

Andrew Stone <andrew@stone.com> is the chief executive haquer at Stone Design Corp <http://www.stone.com/> and divides his time between raising children, llamas & cane and writing applications for Mac OS X and playing with Darwin.

subclasses, and they form a hierarchy that is rooted in the window's content view, the root enclosing view which contains everything in the window except the window controls and shadows. Because you cannot use IB to connect to this content view directly, you'll need to explicitly create a view which contains all of the user interface items that belong in the drawer. Select all the items and choose Layout -> Group in Box. Bring up the Inspector, and choose the Attributes pop-up menu item. Click on "No Title" and the no border icon - now you have an invisible containing view. If you click on the smaller drawer window created when you added the drawer/window combination, you see just such an invisible NSBox. Be sure to add your drawer components inside of this box by double-clicking it before dragging on new user interface elements.

Interface Builder allows you to specify the preferred edge from which the drawer should expand with the NSDrawer Inspector's Attributes sub-panel. For full control of the drawer's appearance and position, you may have to actually write a few lines of code, because some functionality is not yet fully exposed in Interface Builder. As of DP4, you need to set the drawer's delegate and size constraints programmatically. You can control the maximum and minimum size of the drawer, as well as the leading and trailing offsets. On a side mounted window, the leading offset is the height difference between the top of the drawer, and the top of the parent window's content view. Likewise, the trailing offset is the difference between the bottom of the drawer and the bottom of the parent window. All of the size constraints are mostly hints because there may be conflicting parameters.

Now that the drawer is configured, all that is required is to provide the user a means of opening and closing it. You need to have a button on the main window or a menu item which will expand the drawer when closed, and close it when open, simultaneously adjusting the text and/or icon on the button to synchronize with the drawer's state. Because the drawer can be in the act of opening or closing, you might want your button to do something only if it is actually closed or open, and just ignore clicks if the drawer is still animating between the open and closed state.

// given an instance variable "drawer" for the NSDrawer and the sender is the button:

```
- (void)openOrCloseDrawer:(id)sender {
    if ([drawer state] == NSDrawerClosedState) {
  // tell the drawer to begin the opening animation:
    [drawer open:sender];
  // remember, not everyone speaks English! Code internationally:
    [sender setTitle:NSLocalizedStringFromTable(@"Less
Options",@"CoolApp",@"title of drawer open and close button
when the drawer is open")];
    [sender setImage:[NSImage imageNamed:@"OpenDrawer"];
    } else if ([drawer state] == NSDrawerOpenState) {
    [drawer close:sender];
    [sender setTitle:NSLocalizedStringFromTable(@"More
Options",@"CoolApp",@"title of drawer open and close button
when the drawer is closed")];
    [sender setImage:[NSImage imageNamed:@"CloseDrawer"];
    }
  // if it's in an opening or closing state, we'll ignore the click
}
```

As for initializing the drawer, you might want to do some of the following in the method that gets called by any object in an IB nib file after all the outlets are initialized, awakeFromNib:

```
- (void)awakeFromNib
{
   [drawer setLeadingOffset:10.];
   [drawer setTrailingOffset:40.];
   [drawer setContentSize:[rightBox frame].size];
   [drawer close:self];
   [drawer setDelegate:self];
   ...
```

Secret Disclosures

When a drawer is inappropriate because of size, backwards compatibility, or design issues, the classic disclosure view comes in handy. When disclosing additional user interface elements, the programmer is responsible for resizing the window and making sure everything fits correctly. All of this can be done easily using the technique of the invisible box as the top level container of the items in the standard window, and another invisible box as the top level container of the additional items which are presented when the window is expanded. The two most typical configurations are windows which expand to the right, and windows which expand below. We'll look at the expand to the right case first since it is simpler, because it does not involve moving the origin of the window. The underlying window display mechanism in the AppKit will automatically handle the cases where expanding the window would place part of the window off screen.

To understand the automatic resizing behavior of views (autosizing), it is helpful to look at InterfaceBuilder's Autosizing interface element of the Size Inspector. Each of the 6 possible stretch behaviors are represented graphically with rods and springs. A rod means "leave this dimension static" and a spring means "let this dimension fluctuate with the changing size of the window". <<StretchTheObject.tiff In this case, the object stretches and shrinks to fit into its containing view>> <<LeaveObjectRelativeToLowerRight.tiff Here, the object will remain in relative position to the lower right of its containing view.>>

Of course, you can set these all programmatically using NSView's -setAutosizing:(int)mask method by or'ing together the vertical and horizontal stretching behaviors: NSViewNotSizable, NSViewMinXMargin, NSViewWidthSizable, NSViewMaxXMargin, NSViewMinYMargin, NSViewHeightSizable andNSViewMaxYMargin.

Usually, you want the main items in the window to be resized when the user resizes the window - for example, a scrollable text view. In order that the disclosure view maintains the correct size whether it's showing or not, we'll approach the problem by always leaving the extra box in the window. When the extra box is hidden, the window will clip the box, when it's revealed, the window will be resized to contain it. This solves two problems: one, the extra items will be correctly freed when the window is released regardless of whether the extra items are showing, and two, the extra items will be correctly resized when the window is resized, even if they are not currently exposed.

In the following example, we have subclassed NSWindowController and placed the logic of resizing in the subclass controller. In InterfaceBuilder, the autosizing of the elements has been established, and we honor these settings by noting them at the beginning, and resetting them after resizing the window.

The button is set to be a two state "toggle" button, and we assign the images inside InterfaceBuilder in the Icon and Alternate Icon fields. By changing the state of the button, the icon automatically changes. This works with text as well by assigning an alternate title to the button, however, a down facing triangle image when the window is collapsed but can be expanded downward and an upward facing triangle image when it is expanded but can be collapsed works well.

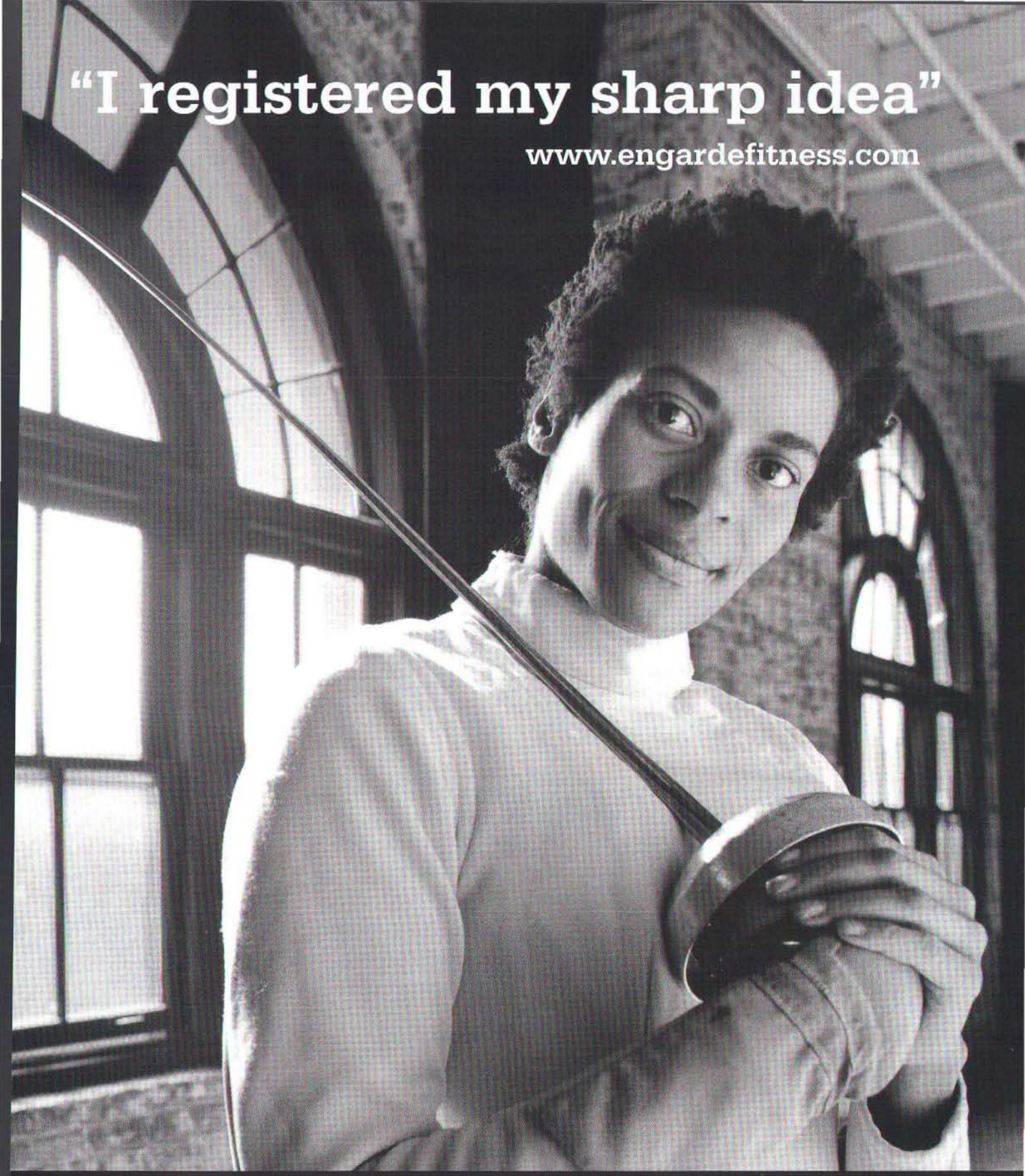
```
// given: the controls to be displayed when the window expands are all inside
// an invisible NSBox named rightBox. The main controls are all inside a box named
// leftBox. The two boxes are laid out side by side in the window to fill the window's
// contentView. Inside the left box, near the upper right is the two-state button whose
// target is the window controller with the action moreOrLessToTheRightAction:.

- (void)moreOrLessToTheRightAction:(id)sender
{
    NSWindow *win = [self window];
    NSRect winFrame = [win frame];
    NSRect rightFrame = [rightBox frame];

// get the original settings for reestablishing later:
    int leftMask = [leftBox autoresizingMask];
    int rightMask = [rightBox autoresizingMask];

// toggle the state
    int stateToSet = 1 - [sender tag];

// set the boxes to not automatically resize when the window resizes:
    [leftBox setAutoresizingMask:NSViewNotSizable];
    [rightBox setAutoresizingMask:NSViewNotSizable];

  // if the button's state is 1, then stateToSet == 0, let's collapse:
    if (stateToSet == 0) {
     // reduce the desired size by the width of the right box:
         winFrame.size.width -= NSWidth(rightFrame);
    } else {
     // increase the desired width by the width of the right box:
         winFrame.size.width += NSWidth(rightFrame);
     }

  // change the state of the button
    [sender setState:stateToSet];
    [sender setTag:stateToSet];

  // resize the window and display:
    [win setFrame:winFrame display:YES];

  // reset the boxes to their original autosize masks:
    [leftBox setAutoresizingMask:leftMask];
    [rightBox setAutoresizingMask:rightMask];
}
```

Adding a disclosure view which expands the window below is slightly more tricky because we'll have to move the origin of the window as we increase or decrease the height of the window so that the window keeps the title bar in the same location. Moreover, since origins begin at the bottom and move to positive Y upwards, we'll have to move the origins of the boxes as well.

```
- (IBAction)moreOrLessDownAction:(id)sender {
    NSWindow *win = [self window];
    NSRect winFrame = [win frame];

// we'll need to know the size of both boxes in this case:
    NSRect topFrame = [topBox frame];
    NSRect bottomFrame = [bottomBox frame];

// get the original settings for reestablishing later:
    int topMask = [topBox autoresizingMask];
    int bottomMask = [bottomBox autoresizingMask];

// toggle the state
    int stateToSet = 1 - [sender tag];

// set the boxes to not automatically resize when the window resizes:
    [topBox setAutoresizingMask:NSViewNotSizable];
    [bottomBox setAutoresizingMask:NSViewNotSizable];

  // if the button's state is 1, then stateToSet == 0, collapse it:
    if (stateToSet == 0) {
    // adjust the desired height and origin of the window:
        winFrame.size.height -= NSHeight(bottomFrame);
        winFrame.origin.y += NSHeight(bottomFrame);
     // adjust the origin of the bottom box well below the window:
        bottomFrame.origin.y = -NSHeight(bottomFrame);
     // begin the top box at the bottom of the window
        topFrame.origin.y = 0.0;
    } else {
    // stack the boxes one on top of the other:
        bottomFrame.origin.y = 0.0;
        topFrame.origin.y = NSHeight(bottomFrame);

    // adjust the desired height and origin of the window:
        winFrame.size.height += NSHeight(bottomFrame);
        winFrame.origin.y -= NSHeight(bottomFrame);
    }

  // adjust locations of the boxes:
    [topBox setFrame:topFrame];
    [bottomBox setFrame:bottomFrame];

  // change the state of the button to reflect new arrangement:
    [sender setState:stateToSet];
    [sender setTag:stateToSet];

  // resize the window and display:
    [win setFrame:winFrame display:YES];

  // reset the boxes to their original autosize masks:
    [topBox setAutoresizingMask:topMask];
    [bottomBox setAutoresizingMask:bottomMask];
}
```

Conclusion

With drawers and disclosure views, you have the tools and techniques to present simple and elegant interfaces, with more features available at the click of a button. InterfaceBuilder can provide almost all of the support necessary to fully implement drawers, and with just a few lines of code, your applications can take advantage of the powerful new features of Cocoa. MT



Sharon Monplaisir, Olympic fencer and entrepreneur, registered her domain name to put her business online at register.com. Visit us at www.register.com or call us at 1-800-7-WWW-NET, and we'll help you register your domain name right over the phone.

register.com™

the *first* step on the web℠

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

Busy Beavers

Before we get to this month's Challenge, I have to confess being a little distracted. No, not because the annual holiday up at the lake is just a few days away, although I'll also confess that the prospect of a couple of weeks away from the Real Job is most appealing. No, the distraction is because UPS just delivered another addition to the family of computers at the Boonstra household. The most recent additions have been iMacs for the Junior members of the family, but this one is for Me. A new G4. No, not one of the new dual-processor models introduced by Apple at JavitsWorld. (Those of us in Boston cannot acknowledge use of the term MacWorld for anything on the Right Coast that doesn't happen in Bean Town.) Dual processors might mean something to those PhotoShop users among you, but they don't do much for the Rest of Us until Mac OS X comes along. No, the new addition is one of those now-obsolete single-processor G4-500 models that have (finally) dropped a little in price. As those of you who participate in the Challenge contests know, I've been limping along with an old 8500, enhanced over the years with a faster 604e, then a dual 604e upgrade (BeOS, oh BeOS, wherefore art thou BeOS?), and finally with a G3 board. Several readers have asked in the past about whether AltiVec technology could be used in the Challenge, but, sadly, I didn't have a G4 to use in the evaluation. A problem now rectified, or at least it will be once I complete the file transfers proceeding even as I write.

Now that you all know about my new toy, we can get on to the business at hand. This month's problem was suggested by F. C Kuechmann, who earns two Challenge points for the suggestion. Your Challenge this month is to create a Busy Beaver Turing Machine and write a program that simulates its execution.

The Busy Beaver problem was invented in the early 1960s by Tibor Rado of Ohio State University. He asked the following question about 2-symbol Turing machines: what is the largest number of 1s that a Turing machine with n states could write to a tape initially filled with 0s. That "busy beaver" number, or BB(n), has some interesting properties. For example, by reasoning about the Halting Problem, one can show that BB(n) grows faster than any computable sequence.

An internet search shows that the Busy Beaver problem continues to attract interest. Until 1985, the largest value for a 5-state busy beaver produced 501 1s. Then George Uhing found a 5-state machine that produced 1915 1s before halting. And in 1987, Heiner Marxen (and Jürgen Buntrock showed that BB(5) is at least 4098.

For reference, you can start with the following URLs: Marxen's page at <http://www.drb.insel.de/~heiner/BB/index.html>, and <http://grail.cba.csuohio.edu/~somos/bb.html>

The prototype for the code you should write is:

```
typedef unsigned long ulong;

typedef enum {kMoveLeft=-1,kHalt=0, kMoveRight=1} MoveDir;

typedef struct TMRule {   /* Turing Machine rule */
  ulong oldState;         /* this rule applies when the machine state is oldState */
  Boolean inputSymbol;    /*  and the current input symbol is inputSymbol */
  ulong newState;         /* set current state to newState when this rule fires */
  Boolean outputSymbol;   /* write outputSymbol to tape when this rule fires */
  char moveDirection;     /* kMoveLeft, kMoveRight, or kHalt */
} TMRule;

ulong /* return number of rules */ BusyBeaver5(
  TMRule theTMRules[],
    /* preallocated storage, return the rules for your BB machine */
);

Boolean /* return true for success */ RunTuringMachine(
  TMRule theTMRules[],
    /* preallocated storage, return the rules for your BB machine */
  ulong numberOfTMRules,
    /* the number of rules in theTMRules */
  ulong numBytesInHalfTape,
    /* half-size of the "infinite" Turing Machine tape */
  unsigned char *tmTape,
    /* pointer to preallocated Turing Machine tape storage */
    /* Each byte contains 8 tape symbols, each symbol is 0 or 1. */
    /* The tape extends from tmTape[-numBytesInHalfTape] to
                                    tmTape[numBytesInHalfTape -1] */
    /* Tape position 0 is (tmTape[0] & 0x80),
       tape position 1 is (tmTape[0] & 0x40)
       tape position -1 is (tmTape[-1] & 0x01), etc. */
  ulong *numberOf1sGenerated,
    /* return the number of 1s placed on the tape */
  ulong *numberOfRulesExecuted
  /* return the number of rules executed when running BB, including the halt rule */

);
```

The first thing you need to do is to select the 5-state Busy Beaver Turing Machine that you will simulate in your RunTuringMachine routine. Since scoring is based on how busy your beaver is, that is, on how many 1s it produces on the simulated Turing Machine tape, you want to give some careful thought to this selection. This Turing Machine should returned by your BusyBeaver5 using the TMRule data structure. BusyBeaver5 may return a hard-coded Turing Machine; it does not need to identify the busy beaver at run time.

My test code will then provide the output of BusyBeaver5 to your RunTuringMachine routine, which should simulate the execution of the input Turing Machine. RunTuringMachine will be provided with a blank (zero-filled) tape tmTape that is 2*numBytesInHalfTape in size. The "read head" of the Turing Machine is initially positioned over position [0] of the tape. On

exit, tmTape should contain the output of the Turing Machine being simulated. In addition, you should return in the appropriate output parameters the number of 1s on the output tape and the number of state transitions that occurred during your execution of the Turing Machine. RunTuringMachine should return TRUE if it was able to successfully execute the Turing Machine, and FALSE if it failed for some reason, such as running out of simulated tape. (It is not my intention to provide a simulated tape that is too short, but your code should fail gracefully if that happens during testing.)

RunTuringMachine must provide a general Turing Machine simulation, not dependent on the Busy Beaver problem or on the content of the initial input tape. I may choose to verify correctness of your RunTuringMachine code against other input besides that produced by BusyBeaver5.

The winner will be the solution that identifies the 5-state Busy Beaver generating the most 1s on the output tape. Among solutions with equal numbers of 1s, the solution that produces the output in the fewest number of Turing Machine steps will be the winner. And, for solutions that produce the same output in the same number of steps, the winner will be the solution that executes the Turing Machine in the least execution time. While my hope is that one of you might break new ground in the field of busy beaver research, my expectation is that the winning solution will be determined by the execution time criterion.

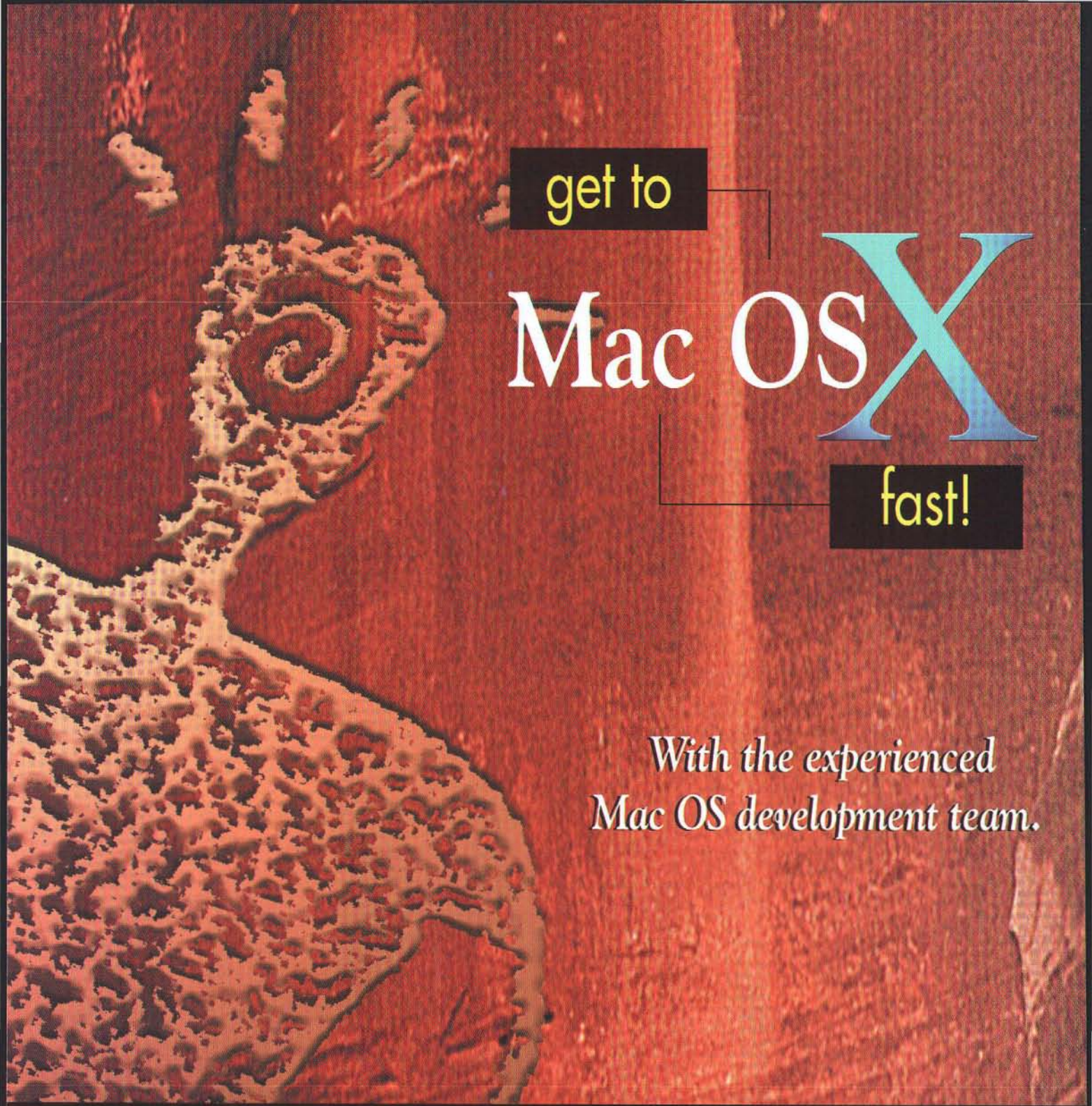
This will be a native PowerPC Challenge, using the CodeWarrior Pro 5 environment. Solutions may be coded in C, C++, or Pascal. As is our tradition for the September Column, we'll also allow solutions that are completely or partially coded in assembly language. And, yes, this time you can take advantage of the AltiVec features of the G4.

Three Months Ago Winner

Congratulations to **Willeke Rieken (The Netherlands)** for submitting the winning solution to the June Rub*k Rotation Programmer's Challenge. Readers may recall that the Rub*k Rotation Challenge required contestants to display an image of the famous puzzle and respond to commands to rotate the entire cube or individual cube faces. Scoring was based on correctness of the solution, in this case the smoothness of the displayed rotations, and on execution time.

The fact that Willeke was the only person to submit an entry does not detract from his solution in the slightest, although it did significantly increase his chances of winning. (You can't win if you don't play!) Willeke elected to use QuickDraw3D to implement his solution, motivated by a desire to gain some experience with the QD3D API. His code creates 26 individual cubies (the center cubie is never visible) using the AddCubie routine. Although it might look like a lot of work to set up the cube, the effort pays off in the simplicity with which one can rotate the cube (RotateCube), turn a face of the cube (QuarterTurn), and draw the entire cube (DrawCube), regardless of orientation.

With only one entry, I'll omit the usual table describing the parameters of the solution, and simply observe that this victory



Red Rock Software specializes in the Macintosh software development. Red Rock Software's team of senior engineers averages over 10 years of development experience on the Macintosh platform. Our expertise and experience has gained us the trust of companies like *Apple Computer, Iomega Corporation and Sorenson Media.*

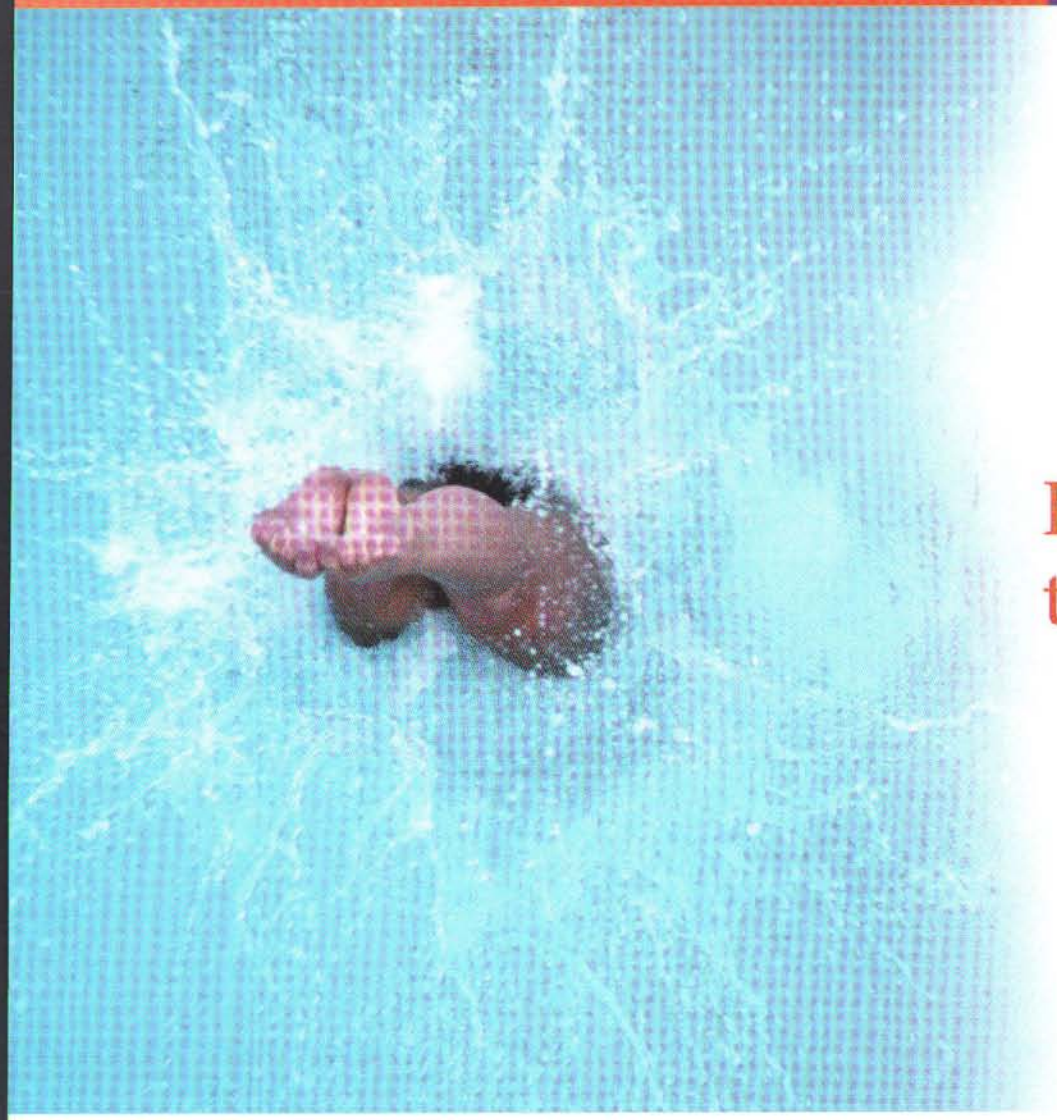
If you are looking to get your product compatible with Mac OS X quickly and with extreme quality, let our experts help.

- Aqua Interface Implementation
- OS X Carbon Porting
- OS X Native Cocoa Application Development

RED ROCK
software

call Red Rock at 888.689.3038
or visit us online at www.redrocksw.c

OS X Development



Ready to take the plunge?

Okay, everyone into the Aqua! What? Can't swim? No problem. Just climb on our back. We've been to the bottom of Apple's new Mac OS X, far beneath its cool Aqua interface. Porting to Carbon. Diving into Cocoa. Firing up PowerPlant. And developing KEXTs, NKEs, and IOKit drivers. If you're working on an OS X project, we can support you at every level.

A deep pool of talent

We founded our firm on the Mac a decade ago. It's in our DNA. Which is why clients such as 3dfx, Apple, and Lexmark come to us for assistance. We specialize in full-service commercial software development and we thrive on challenging projects. The ones that make your head spin and your staff suck for air.

Contact us soon. We'll help you make a splash in the market —cannonball size.

Critical Path Software

711 SW Alder Street, 4th Floor, Portland, OR 97205 503.222.2922, www.criticalpath.com
Portland • San Diego • Tokyo

Robosoft Technologies

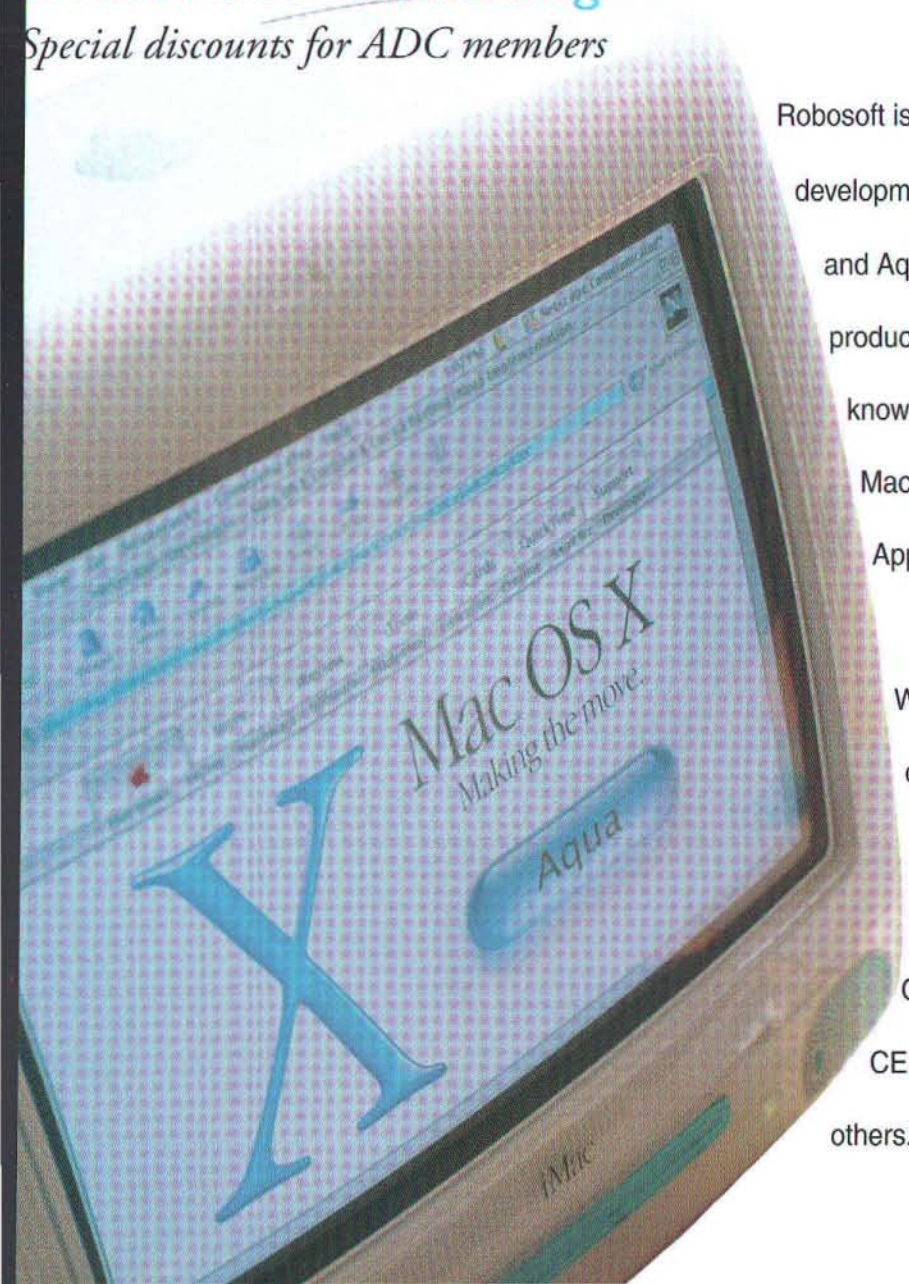
www.robosoftin.com

partners in product development

Indian mind power@ an unbeatable price

Carbon Porting
Aqua Implementation
Windows to Mac Porting

Special discounts for ADC members



Robosoft is an India based specialist in Mac product development, Windows to Mac porting, Carbon porting and Aqua implementation. Our engineers have a strong product development experience with a thorough knowledge of C/C++, Java,Mac OS Toolbox, Macsbug, QuickTime, Carbon API and PowerPlant Application Framework.

We have a proven track record for product development out sourcing and delivering the projects on time. Our clients include a veritable who's who in the industry - Apple, Adobe, Quark, FileWave, Versaware, Veon, NetJumper, CEI, The Anderson Group and eCapital, among others. To know us better, visit our website

>>>>>

info@robosoftin.com

- [ ] Eat your vegetables.
- [ ] Exercise every day.
- [x] Port to Mac OS X.
- [ ] Call your mom.

All of these are good for you. We can make one of them easy.

Since 1989, The Omni Group has worked with the technologies that have been refined into Mac OS X.

Moving to Mac OS X is a big step for your company, and you need consultants who can help you both plan how best to make the transition and follow whatever path you choose.

That's been our business for years. No matter what kind of product you have, we can get it up under OS X, fast:

- Real games: We ported id's Doom and Quake games to NEXTSTEP and OS X. Quake 2 took us a week.
- Big apps: We ported Adobe's FrameMaker to NEXTSTEP and Sun's Concurrence to OpenStep/Solaris.
- Big libraries: We ported the Oracle 8 client libraries which Apple ships today in OS X Server.
- Serious drivers: We ported 3dfx's Glide and wrote Voodoo2 and Rendition drivers for OS X Server. We've written new mouse drivers for OS X Server and joystick drivers for OpenStep.
- New apps: We wrote OmniWeb, the only native OS X web browser, and OmniPDF, the native Acrobat viewer for OS X.

Mac OS X is what we do. Let us help you do it, too.

THE OMNI GROUP

2707 Northeast Blakeley Street
Seattle, Washington 98105-3118
www.omnigroup.com/consulting

sales@omnigroup.com
800.315.OMNI x201
206.523.4152 x201

Get the power and experience you need from

PROSOFT
engineering, inc.

With over thirteen years in providing programming service to the Mac community. Prosoft is committed in delivering the ultimate quality software products and service.

Got OS X?

Our Engineers have extensive knowledge in:

- Macintosh PPC Drivers, Shims and Extensions
- Macintosh Power Plant Applications, Mac OS X applications and drivers
- Carbon application porting for Mac OS X
- Multimedia and QuickTime Applications
- Unix/Linux Console Applications, including Unix Solaris, xBSD Drivers, and Linux Drivers

Other areas of expertise

- Windows 95, 98
- Window CE
- Windows NT, 2000
- Unix / Linux
- Java
- Embedded OS
- RDBMS
- Network Consulting

www.prosofteng.com • tel: (925) 426-6100 • sales@prosofteng.co

vaults Willeke into 4th place in the overall Challenge standings. And remember, you can't win if you don't ...., oh, I'm repeating myself.

Top Contestants

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 10 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	245	10.	Downs, Andrew	12
2.	Saxton, Tom	126	11.	Jones, Dennis	12
3.	Maurer, Sebastian	78	12.	Day, Mark	10
4.	Rieken, Willeke	65	13.	Duga, Brady	10
5.	Boring, Randy	50	14.	Fazekas, Miklos	10
6.	Shearer, Rob	47	15.	Murphy, ACC	10
7.	Taylor, Jonathan	26	16.	Selengut, Jared	10
8.	Brown, Pat	20	17.	Strout, Joe	10
9.	Heithcock, JG	16			

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Willeke's winning Rub*k Rotation solution:

RubikRotation.c
Copyright © 2000
Willeke Rieken

```
/*
    draws (a simplified version of) Rubik's cube and animates
    rotations of the cube and of the faces of the cube.
    I'm using QD3D because I never used it and it seemed
    more fun than a diy method and drowning in sin and cos
    and I still think it is.

    the model object for the cube consists of 26 group objects
    for each cubie and a rotation object. the rotation object
    contains all previous rotations of the whole cube added together.
    each cubie object contains a box object and a rotation object.
    a cubie rotation object contains all previous rotations of the
    cubie that was caused by rotating a face of the cube.

    during rotation of the cube an extra rotation object is
    submitted. after the rotation the rotation object of the cube
    is adjusted.
    during rotation of a face an exta rotation object is added to
    every cubie in the face. after the rotation the extra rotation
    object is removed and the rotation object of the cubie is adjusted.

    references to the rotation object of the cube and to the cubie objects
    are kept in globals.
*/

#include <QD3D.h>
#include <QD3DDrawContext.h>
#include <QD3DRenderer.h>
#include <QD3DShader.h>
#include <QD3DCamera.h>
#include <QD3DLight.h>
#include <QD3DGeometry.h>
#include <QD3DGroup.h>
#include <QD3DMath.h>
#include <QD3DTransform.h>
#include <QD3DView.h>
#include <QD3DAcceleration.h>
#include <QD3DErrors.h>

#include "RubikRotation.h"

TQ3ViewObject gView; // the view for the scene
TQ3StyleObject  gInterpolation;
        // interpolation style used when rendering
TQ3StyleObject  gBackFacing;
        // whether to draw shapes that face away from the camera
TQ3StyleObject  gFillStyle;
        // whether drawn as solid filled object or decomposed to components
TQ3GroupObject  gCubeModel; // the cube
TQ3GroupObject  gCubies[3][3][3];
        // the cubies
TQ3TransformObject gCubeRotation;
        // cumulation of every rotation of the whole cube until now
TQ3TransformObject gTempCubeRotation;
        // used during rotation of the cube
float  gStepSize;

static TQ3DrawContextObject MyNewDrawContext(CWindowPtr
theWindow)
// create context
{
  TQ3DrawContextData myDrawContextData;
  TQ3MacDrawContextData  myMacDrawContextData;
  TQ3ColorARGB  clearColor;
  TQ3DrawContextObject myDrawContext ;

  // Set the background color
  clearColor.a = 1.0;
  clearColor.r = 1.0;
  clearColor.g = 1.0;
  clearColor.b = 1.0;

  // Fill in draw context data
  myDrawContextData.clearImageMethod = kQ3ClearMethodWithColor;
  myDrawContextData.clearImageColor = clearColor;
  myDrawContextData.paneState = kQ3False;
  myDrawContextData.maskState = kQ3False;
  myDrawContextData.doubleBufferState = kQ3True;
  myMacDrawContextData.drawContextData = myDrawContextData;
  myMacDrawContextData.window = theWindow;
  myMacDrawContextData.library = kQ3Mac2DLibraryNone;
  myMacDrawContextData.viewPort = 0;
  myMacDrawContextData.grafPort = 0;

  // Create draw context
  myDrawContext = Q3MacDrawContext_New(&myMacDrawContextData) ;
  return myDrawContext ;
}

static TQ3CameraObject MyNewOrthographicCamera(CWindowPtr
theWindow, short cubeWidth)
// create orthographic camera
{
  TQ3OrthographicCameraData orthographicData;
  TQ3CameraObject camera;
  TQ3Point3D  from = {0.0, 1.5, 7.0};
  TQ3Point3D  to = {0.0, 0.0, 0.0};
  TQ3Vector3D up = {0.0, 1.0, 0.0};

  orthographicData.cameraData.placement.cameraLocation = from;
  orthographicData.cameraData.placement.pointOfInterest = to;
```

```
  orthographicData.cameraData.placement.cameraLocation =
from;
  orthographicData.cameraData.placement.pointOfInterest = to;
  orthographicData.cameraData.placement.upVector = up;
  orthographicData.cameraData.range.hither = 1.0;
  orthographicData.cameraData.range.yon = 1000.0;
  orthographicData.cameraData.viewPort.origin.x = -1.0;
  orthographicData.cameraData.viewPort.origin.y = 1.0;
  orthographicData.cameraData.viewPort.width = 2.0;
  orthographicData.cameraData.viewPort.height = 2.0;

  // calculate view plane, size of the cube is 3.0 in QD3Points
  orthographicData.left = -1.5 *
      ((float)(theWindow->portRect.right -
               theWindow->portRect.left)) /
          (float)(cubeWidth + 1);
  orthographicData.top = orthographicData.left;
  orthographicData.right = -orthographicData.left;
  orthographicData.bottom = orthographicData.right;

  camera = Q3OrthographicCamera_New(&orthographicData);
  return camera;
}

static TQ3CameraObject MyNewViewPlaneCamera(CWindowPtr
theWindow, short cubeWidth)
{
// create perspective camera
  TQ3ViewPlaneCameraData viewPlaneData;
  TQ3CameraObject camera;
  TQ3Point3D from = {0.0, 0.0, 7.0};
  TQ3Point3D to = {0.0, 0.0, 1.5};
  TQ3Vector3D up = {0.0, 1.0, 0.0};

  viewPlaneData.cameraData.placement.cameraLocation = from;
  viewPlaneData.cameraData.placement.pointOfInterest = to;
  viewPlaneData.cameraData.placement.upVector = up;
  viewPlaneData.cameraData.range.hither = 1.0;
  viewPlaneData.cameraData.range.yon = 1000.0;
  viewPlaneData.cameraData.viewPort.origin.x = -1.0;
  viewPlaneData.cameraData.viewPort.origin.y = 1.0;
  viewPlaneData.cameraData.viewPort.width = 2.0;
  viewPlaneData.cameraData.viewPort.height = 2.0;

  // calculate view plane, size of the cube is 3.0 in QD3Points
  viewPlaneData.viewPlane = 5.5;
  viewPlaneData.halfWidthAtViewPlane = 1.5 *
      ((float)(theWindow->portRect.right -
               theWindow->portRect.left)) /
          (float)(cubeWidth + 1);
  viewPlaneData.halfHeightAtViewPlane =
      viewPlaneData.halfWidthAtViewPlane;
  viewPlaneData.centerXOnViewPlane = 0.0;
  viewPlaneData.centerYOnViewPlane = 0.0;

  camera = Q3ViewPlaneCamera_New(&viewPlaneData);
  return camera;
}

static TQ3GroupObject MyNewAmbientOnlyLights()
{
  TQ3GroupObject  myLightList;
  TQ3LightData  myLightData;
  TQ3LightObject  myAmbientLight;
  TQ3ColorRGB whiteLight = {1.0, 1.0, 1.0};

  // Set up light data for ambient light.
  myLightData.isOn = kQ3True;
  myLightData.color = whiteLight;

  // Create ambient light.
  myLightData.brightness = 1.0;
  myAmbientLight = Q3AmbientLight_New(&myLightData);

  // Create light group and add each of the lights into the  group.
  myLightList = Q3LightGroup_New();
  Q3Group_AddObject(myLightList, myAmbientLight);
  Q3Object_Dispose(myAmbientLight) ;
  return myLightList;
}

static TQ3GroupObject MyNewLights()
{
  TQ3GroupObject  myLightList;
  TQ3LightData  myLightData;
  TQ3PointLightData myPointLightData;
  TQ3DirectionalLightData  myDirectionalLightData;
  TQ3LightObject  myAmbientLight, myPointLight, myFillLight;
  TQ3Point3D  pointLocation = {-10.0, 0.0, 10.0};
  TQ3Vector3D fillDirection = {10.0, 0.0, 10.0};
  TQ3ColorRGB whiteLight = {1.0, 1.0, 1.0};

  // Set up light data for ambient light.
  // This light data will be used for point and fill light also.
  myLightData.isOn = kQ3True;
  myLightData.color = whiteLight;

  // Create ambient light.
  myLightData.brightness = 0.25;
  myAmbientLight = Q3AmbientLight_New(&myLightData);

  // Create point light.
  myLightData.brightness = 1.0;
  myPointLightData.lightData = myLightData;
  myPointLightData.castsShadows = kQ3False;
  myPointLightData.attenuation = kQ3AttenuationTypeNone;
  myPointLightData.location = pointLocation;
  myPointLight = Q3PointLight_New(&myPointLightData);

  // Create fill light.
  myLightData.brightness = 0.2;
  myDirectionalLightData.lightData = myLightData;
  myDirectionalLightData.castsShadows = kQ3False;
  myDirectionalLightData.direction = fillDirection;
  myFillLight =
Q3DirectionalLight_New(&myDirectionalLightData);

  // Create light group and add each of the lights into the group.
  myLightList = Q3LightGroup_New();
  Q3Group_AddObject(myLightList, myAmbientLight);
  Q3Group_AddObject(myLightList, myPointLight);
  Q3Group_AddObject(myLightList, myFillLight);

  Q3Object_Dispose(myAmbientLight);
  Q3Object_Dispose(myPointLight);
  Q3Object_Dispose(myFillLight);

  return myLightList;
}

static TQ3ViewObject MyNewView(CWindowPtr theWindow, short
cubeWidth)
{
  TQ3ViewObject myView;
  TQ3DrawContextObject myDrawContext;
  TQ3RendererObject  myRenderer;
  TQ3CameraObject myCamera;
  TQ3GroupObject  myLights;

  myView = Q3View_New();

  // Create and set draw context.
  myDrawContext = MyNewDrawContext(theWindow);
  Q3View_SetDrawContext(myView, myDrawContext);
  Q3Object_Dispose(myDrawContext) ;

  // Create and set renderer.
  // use the interactive software renderer
  myRenderer =
    Q3Renderer_NewFromType(kQ3RendererTypeInteractive);
  Q3View_SetRenderer(myView, myRenderer);
  // these two lines set us up to use the best possible renderer,
  // including  hardware if it is installed.
  Q3InteractiveRenderer_SetDoubleBufferBypass(myRenderer,
      kQ3True);
  Q3InteractiveRenderer_SetPreferences(myRenderer,
      kQAVendor_BestChoice, 0);
  /* for software renderer, without hardware accelleration, replace with:
  Q3InteractiveRenderer_SetPreferences(myRenderer, kQAVendor_Apple,
      kQAEngine_AppleSW);
  */
```

```
  Q3Object_Dispose(myRenderer);

  // Create and set camera.
  myCamera = MyNewViewPlaneCamera(theWindow, cubeWidth);
  /* for an orthographic camera, replace with:
  myCamera = MyNewOrthographicCamera(theWindow, cubeWidth);
  */
  Q3View_SetCamera(myView, myCamera);
  Q3Object_Dispose(myCamera) ;

  // Create and set lights.
  myLights = MyNewAmbientOnlyLights();
  /* for better looking lights, replace with:
  myLights = MyNewLights();
  */
  Q3View_SetLightGroup(myView, myLights);
  Q3Object_Dispose(myLights);

  return myView;
}

static void DrawCube()
{
  TQ3ViewStatus myStatus;
  Q3View_StartRendering(gView);
  do
  {
    Q3Style_Submit(gInterpolation, gView);
    Q3Style_Submit(gBackFacing, gView);
    Q3Style_Submit(gFillStyle, gView);
    if (gTempCubeRotation)
      Q3Transform_Submit(gTempCubeRotation, gView);
    Q3DisplayGroup_Submit(gCubeModel, gView);
    myStatus = Q3View_EndRendering(gView);
  } while (myStatus == kQ3ViewStatusRetraverse);
}

static void AddCubie(TQ3GroupObject theGroup, long theX, long
theY, long theZ,
            TQ3ColorRGB *theLeftColor, TQ3ColorRGB
*theRightColor, TQ3ColorRGB *theFrontColor,
            TQ3ColorRGB *theBackColor, TQ3ColorRGB
*theTopColor, TQ3ColorRGB *theBottomColor)
{
  TQ3GeometryObject myBox;
  TQ3BoxData myBoxData;
  TQ3SetObject faces[6];
  TQ3GroupObject aCubie;
  TQ3TransformObject aTransformation;
  TQ3Matrix4x4 aMatrix;
  short face;

  // create a rotation object, it doesn't rotate yet
  // but it will be adjusted after rotating the face
  aCubie = Q3DisplayGroup_New();
  Q3Matrix4x4_SetIdentity(&aMatrix);
  aTransformation = Q3MatrixTransform_New(&aMatrix);
  Q3Group_AddObject(aCubie, aTransformation);
  Q3Object_Dispose(aTransformation);

  // create the box itself
  myBoxData.faceAttributeSet = faces;
  myBoxData.boxAttributeSet = nil;
  myBoxData.faceAttributeSet[0] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[0],
      kQ3AttributeTypeDiffuseColor, theLeftColor);
  myBoxData.faceAttributeSet[1] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[1],
      kQ3AttributeTypeDiffuseColor, theRightColor);
  myBoxData.faceAttributeSet[2] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[2],
      kQ3AttributeTypeDiffuseColor, theFrontColor);
  myBoxData.faceAttributeSet[3] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[3],
      kQ3AttributeTypeDiffuseColor, theBackColor);
  myBoxData.faceAttributeSet[4] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[4],
      kQ3AttributeTypeDiffuseColor, theTopColor);
  myBoxData.faceAttributeSet[5] = Q3AttributeSet_New();
  Q3AttributeSet_Add(myBoxData.faceAttributeSet[5],
      kQ3AttributeTypeDiffuseColor, theBottomColor);
  Q3Point3D_Set(&myBoxData.origin, -1.5 + theX, 0.5 - theY,
      0.5 - theZ);
  Q3Vector3D_Set(&myBoxData.orientation, 0, 1, 0);
  Q3Vector3D_Set(&myBoxData.majorAxis, 0, 0, 1);
  Q3Vector3D_Set(&myBoxData.minorAxis, 1, 0, 0);
  myBox = Q3Box_New(&myBoxData);
  for (face = 0; face < 6; face++)
    if (myBoxData.faceAttributeSet[face] != 0)
      Q3Object_Dispose(myBoxData.faceAttributeSet[face]);
  Q3Group_AddObject(aCubie, myBox);
  Q3Object_Dispose(myBox);
  Q3Group_AddObject(theGroup, aCubie);
  gCubies[theX][theY][theZ] = aCubie;
}

static TQ3GroupObject MyNewModel(const RGBColor cubeColors[6],
      const short cubieColors[6][3][3])
{
  TQ3GroupObject myGroup = 0;
  TQ3ShaderObject myIlluminationShader ;
  TQ3Matrix4x4 aMatrix;
  TQ3ColorRGB Q3CubeColors[6];
  TQ3ColorRGB aGray = {0.25, 0.25, 0.25};
  long face;

  // convert RGBColor to TQ3ColorRGB
  for (face = 0; face < 6; face++)
  {
Q3CubeColors[face].r = (float)cubeColors[face].red / 0xffff;
Q3CubeColors[face].g = (float)cubeColors[face].green / 0xffff;
Q3CubeColors[face].b = (float)cubeColors[face].blue / 0xffff;
  }
  // Create a group for the complete model.
  if ((myGroup = Q3DisplayGroup_New()) != 0)
  {
    // Define a shading type for the group
    // and add the shader to the group
    myIlluminationShader = Q3NULLIllumination_New();
    /* for a better looking cube, replace with
    myIlluminationShader = Q3LambertIllumination_New();
    or
    myIlluminationShader = Q3PhongIllumination_New();
    */
    Q3Group_AddObject(myGroup, myIlluminationShader);
    Q3Object_Dispose(myIlluminationShader);

  // create a rotation object, it doesn't rotate yet
  // but it will be adjusted after rotating the cube
    Q3Matrix4x4_SetIdentity(&aMatrix);
    gCubeRotation = Q3MatrixTransform_New(&aMatrix);
    Q3Group_AddObject(myGroup, gCubeRotation);

    // add boxes for the cubies
      // left top front
    AddCubie(myGroup, 0, 0, 0,
          &Q3CubeColors[cubieColors[kLeft][2][0]], &aGray,
          &Q3CubeColors[cubieColors[kFront][0][0]],
      &aGray, &Q3CubeColors[cubieColors[kUp][0][2]], &aGray);
      // middle top front
    AddCubie(myGroup, 1, 0, 0,
    &aGray, &aGray, &Q3CubeColors[cubieColors[kFront][1][0]],
      &aGray, &Q3CubeColors[cubieColors[kUp][1][2]], &aGray);
      // right top front
    AddCubie(myGroup, 2, 0, 0,
          &aGray, &Q3CubeColors[cubieColors[kRight][0][0]],
          &Q3CubeColors[cubieColors[kFront][2][0]],
      &aGray, &Q3CubeColors[cubieColors[kUp][2][2]], &aGray);

      // left top middle
    AddCubie(myGroup, 0, 0, 1,
    &Q3CubeColors[cubieColors[kLeft][1][0]], &aGray, &aGray,
      &aGray, &Q3CubeColors[cubieColors[kUp][0][1]], &aGray);
      // middle top middle
    AddCubie(myGroup, 1, 0, 1,
          &aGray, &aGray, &aGray,
      &aGray, &Q3CubeColors[cubieColors[kUp][1][1]], &aGray);
      // right top middle
    AddCubie(myGroup, 2, 0, 1,
    &aGray, &Q3CubeColors[cubieColors[kRight][1][0]], &aGray,
      &aGray, &Q3CubeColors[cubieColors[kUp][2][1]], &aGray);
```

AS A PROGRAMMER OR SYSADMIN, YOU'VE GOT BETTER THINGS TO DO THAN FIGHT WITH A WEB SITE

Searchable Databases
Online Stores
Info Baskets
Guestbooks
Banner Ads
Credit Card Transactions
Counters
Password Protection
Surveys
Quizzes
Form Processing
Conditionals
Math
Cookies
Date and Time
File Includes
File Writes
Web Based Email
Classified Ads
Discussion Groups
Web Chat
Affiliate Tracking
I/O Transactions
Δ Time Calculations
PGP Encryption
Token Tracking
Calendars
Random Rotation
Send Mail
E - Cards
Credit Bank

It's time for you to take a look at MGI

MGI, a plug-in to 4D's award-winning server, WebSTAR, is designed to provide functionality to otherwise static web sites, whether for a private intranet or enterprise - class ISP/ASP. MGI was specifically developed to be used by web graphic designers with no scripting or programming experience. If you know HTML, you know MGI. And if you are a programmer, you can learn MGI by the time your pizza is delivered. You can try out MGI for free – right now – without obligation by downloading a fully - functioning demo at :

http://www.pageplanetsoftware.com

page planet software inc

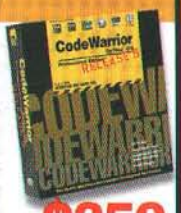
SOFTWARE BUILT WITH YOU IN MIND

PagePlanet Software Inc., 3252 Octavia Street, Raleigh, NC 27606 919.233.1770

The products you need, with the prices and service you deserve... guaranteed.

Power Tools for Programmers!

CodeWarrior Pro 5



CodeWarrior Professional 5 put everything you need for software development at your fingertips: project management tools, text and resource editors, source and class browsers, compilers, linkers, assemblers, and debugger.Release 5 offers such features as RAD for Java, faster compile times, local and remote application debugging, IDE extensibility options and even tighter C/C++ compliance. Additionally, you can create applications for Windows 95/98/NT and Mac OS 8.x and Mac OS X from either host platform. (available in versions hosted on Mac or Windows).

$359

Installer Maker 6.5 10K

Whether you're a developer of high-end, complex applications, simpler utilities, shareware/freeware, an IS manager or an ISP who needs to distribute files quickly and easily, the new InstallerMaker is the complete installation solution for you. Utilizing the power of the new StuffIt Engine, InstallerMaker creates installers faster and smaller than ever, decreasing download time off servers and reducing the number of disks needed to distribute installers. These time and cost savings go straight to your bottom line!

$219

Spotlight

Spotlight is the first Macintosh "Automatic Debugger". It can automatically locate run time errors in your code and display the offending source code line. Unlike similar tools on other platforms Spotlight is easy to use. No source code changes are necessary for application debugging. Spotlight can automatically check for wild pointers, memory leaks, overwrites, underwrites, invalid dereferencing of handles, and even toolbox parameter validity checking -- spotlight knows Macintosh verifying parameters to over 400 toolbox calls.

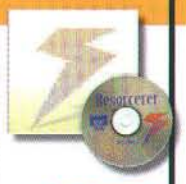
$189

VOODOO Server

VOODOO Server is a version control system for software developers using Metrowerks CodeWarrior under Mac OS. VOODOO Server and the corresponding VOODOO clients (the included CodeWarrior VCS plug-in and the VOODOO Admin application) are designed to offer reliable and robust version control features while minimizing the administrative overhead that usually accompanies version control. If you're a single programmer or managing a team of developers, version control can make or break your project. Do it right, with VOODOO

$79

Resorcerer 2.2



Resorcerer is the only supported general-purpose resource editor for Macintosh. Relied upon by thousands of Mac developers, Resorcerer features a wealth of powerful yet easy-to-use tools for easier, faster, and safer editing of Macintosh data files and resources. Whether you have to parse a picture, debug a data fork, design and try out Balloon Help, create a scripting dictionary, create anti-aliased icons, design and edit a custom resource with 40,000 fields in it, create C source code to run a dialog, or any of hundreds of other resource-related tasks, Resorcerer's magic will quickly save you time and money.

$256

Future BASIC 3

NEW!

One of the most flexible and powerful development environments on the Macintosh today! Easily create programs with the visual program editor, drop into the BASIC editor to define powerful logic with the worlds easiest programming language, or work directly with the Macintosh toolbox. The only BASIC compiler on the market that gives you 100% access to all of the power of the Macintosh toolbox!

$159

and hundreds more!

Page Charmer 2.0 $139

Scripter 2.0 $179

WebTen $349

FaceSpan 3.0 $179

ToolsPlus Lite $99

WebSpice 1,000,000 $89

WebSpice Animations $89

MkLinux $39

PowerKey Rebound $89

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

www.devdepot.com

Master the Web!

WebSTAR Server Suite 4.2

WebSTAR Server Suite is a complete set of powerful and easy-to-use Internet servers for the Mac OS. Effortlessly serve web pages, host email accounts, publish databases, and share files - all with a single application on one Mac! WebSTAR Server Suite is perfect for Internet or Intranet serving, single or multiple sites, small and large businesses - it's power and ease-of-use saves any organization time and money.

$539

CyberGauge 3.0

NEW!

Monitor the bandwidth usage of up to five different machines on your network! Do you need to upgrade your webserver? How hard is your eMail server working? Are you getting all the bandwidth you're paying for? Not only can CyberGauge answer all these questions, new features allow CyberGauge to eMail or page your network device becomes unresponsive r passes a threshold of usage you define – an essential first line of defense for early detection of denial of service attacks and necessity for warning you and tracking quality of ISPs that may have brown outs and shutdowns.

$249

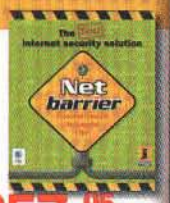
Funnel Web Pro

NEW!

Funnel Web is the ultimate web analysis solution for professionals. Specifically designed for profiling web site usage and monitoring customer usage patterns, Funnel Web is ideal for examining server performance and online effectiveness. Funnel Web can analyze log file formats from any server, WebSTAR, WebTen, even Unix or NT hosted ervers. Discover the most popular pages on your site, track server loads & optimize server performance, profile visitors based on organization, domain name, country, browser, etc. Funnel Web does it all!

$329

NetBarrier



NetBarrier offers a Personal Firewall, Antivandal protection, and Internet Filtering. Protect your machine from intrusions by Internet or AppleTalk. Incorrect passwords and individual actions are logged, you are alerted to hostile actions, and intruders are easily isolated. Internet Filtering allows you to be sure that passwords, credit card numbers, and other sensitive information can never be exported from your computer – the content itself is filtered before any transfer! (Rated 4 mice by Macworld Magazine)

$57.95

...and great hardware solutions!

2 USB PCI Card

Add two USB ports to your older Macintosh. Connect up to 127 devices to the Universal Serial Bus (USB) that is Apple's new standard for desktop connectivity. USB mouse devices, keyboards, joysticks, game controllers, printers, scanners — connect them all to your current computer. Installs in minutes!

$32.95

Dr. Bott Moni Switch ADB or USB

NEW!

Do you need 4 monitors and 4 keyboards for your 4 servers? With Dr. Bott Moni-Switch you can connect a single keyboard and monitor to up to 4 machines at once! A simple flick of a switch directs the video input and keyboard commands to the appropriate CPU! Available in USB and ADB models, with 2 or 4 machine support, and bundles with USB PCI cards so you can mix and match USB and ADB machines with the same Moni-Switch! Great for programmers to do back ground compiles, ideal for server rooms overcrowded with monitors and keyboards!

As low as
$129.95

Macsense USB Full Size Keyboard

Just plug this keyboard into your Mac and start typing! The UKB-600 keyboard from Macsense is designed to get you typing quickly and easily, without any hassle or compatibility worries. It features two tone translucent design, colored to match your favorite flavor of Macintosh. It offers soft touch with positive tactile feedback and build built in USB port on either side of the keyboard. Includes a 5' USB cable and is 100% Macintosh compatible, simply plug and play, as easy as Macintosh!

$44.95

Macsense Internet Sharing Router

Looking to get your whole office online without shelling out thousands of dollars? If so, the XRouter Internet Sharing Hub offers the perfect solution. This amazing Ethernet-to-Ethernet hub connects an entire network of up to 252 users to the Internet using only one ISP account and one Cable or DSL modem!

$199

```
    // left top back
  AddCubie(myGroup, 0, 0, 2,
  &Q3CubeColors[cubieColors[kLeft][0][0]], &aGray, &aGray,
        &Q3CubeColors[cubieColors[kBack][2][0]],
        &Q3CubeColors[cubieColors[kUp][0][0]], &aGray);
    // middle top back
  AddCubie(myGroup, 1, 0, 2,
        &aGray, &aGray, &aGray,
        &Q3CubeColors[cubieColors[kBack][1][0]],
        &Q3CubeColors[cubieColors[kUp][1][0]], &aGray);
    // right top back
  AddCubie(myGroup, 2, 0, 2,
  &aGray, &Q3CubeColors[cubieColors[kRight][2][0]], &aGray,
        &Q3CubeColors[cubieColors[kBack][0][0]],
        &Q3CubeColors[cubieColors[kUp][2][0]], &aGray);

    // left middle front
  AddCubie(myGroup, 0, 1, 0,
        &Q3CubeColors[cubieColors[kLeft][2][1]], &aGray,
        &Q3CubeColors[cubieColors[kFront][0][1]],
        &aGray, &aGray, &aGray);
    // middle middle front
  AddCubie(myGroup, 1, 1, 0,
  &aGray, &aGray, &Q3CubeColors[cubieColors[kFront][1][1]],
        &aGray, &aGray, &aGray);
    // right middle front
  AddCubie(myGroup, 2, 1, 0,
        &aGray, &Q3CubeColors[cubieColors[kRight][0][1]],
        &Q3CubeColors[cubieColors[kFront][2][1]],
        &aGray, &aGray, &aGray);

    // left middle middle
  AddCubie(myGroup, 0, 1, 1,
  &Q3CubeColors[cubieColors[kLeft][1][1]], &aGray, &aGray,
        &aGray, &aGray, &aGray);
    // middle middle middle
  /* invisible
  AddCubie(myGroup, 1, 1, 1,
        &aGray, &aGray, &aGray,
        &aGray, &aGray, &aGray);
  */
    // right middle middle
  AddCubie(myGroup, 2, 1, 1,
  &aGray, &Q3CubeColors[cubieColors[kRight][1][1]], &aGray,
        &aGray, &aGray, &aGray);

    // left middle back
  AddCubie(myGroup, 0, 1, 2,
  &Q3CubeColors[cubieColors[kLeft][0][1]], &aGray, &aGray,
  &Q3CubeColors[cubieColors[kBack][2][1]], &aGray, &aGray);
    // middle middle back
  AddCubie(myGroup, 1, 1, 2,
        &aGray, &aGray, &aGray,
  &Q3CubeColors[cubieColors[kBack][1][1]], &aGray, &aGray);
    // right middle back
  AddCubie(myGroup, 2, 1, 2,
  &aGray, &Q3CubeColors[cubieColors[kRight][2][1]], &aGray,
  &Q3CubeColors[cubieColors[kBack][0][1]], &aGray, &aGray);

    // left bottom front
  AddCubie(myGroup, 0, 2, 0,
        &Q3CubeColors[cubieColors[kLeft][2][2]], &aGray,
        &Q3CubeColors[cubieColors[kFront][0][2]],
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][0][0]]);
    // middle bottom front
  AddCubie(myGroup, 1, 2, 0,
  &aGray, &aGray, &Q3CubeColors[cubieColors[kFront][1][2]],
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][1][0]]);
    // right bottom front
  AddCubie(myGroup, 2, 2, 0,
        &aGray, &Q3CubeColors[cubieColors[kRight][0][2]],
        &Q3CubeColors[cubieColors[kFront][2][2]],
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][2][0]]);

    // left bottom middle
  AddCubie(myGroup, 0, 2, 1,
  &Q3CubeColors[cubieColors[kLeft][1][2]], &aGray, &aGray,
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][0][1]]);
    // middle bottom middle
  AddCubie(myGroup, 1, 2, 1,
        &aGray, &aGray, &aGray,
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][1][1]]);
    // right bottom middle
  AddCubie(myGroup, 2, 2, 1,
  &aGray, &Q3CubeColors[cubieColors[kRight][1][2]], &aGray,
  &aGray, &aGray, &Q3CubeColors[cubieColors[kDown][2][1]]);

    // left bottom back
  AddCubie(myGroup, 0, 2, 2,
  &Q3CubeColors[cubieColors[kLeft][0][2]], &aGray, &aGray,
        &Q3CubeColors[cubieColors[kBack][2][2]], &aGray,
        &Q3CubeColors[cubieColors[kDown][0][2]]);
    // middle bottom back
  AddCubie(myGroup, 1, 2, 2,
        &aGray, &aGray, &aGray,
        &Q3CubeColors[cubieColors[kBack][1][2]], &aGray,
        &Q3CubeColors[cubieColors[kDown][1][2]]);
    // right bottom back
  AddCubie(myGroup, 2, 2, 2,
  &aGray, &Q3CubeColors[cubieColors[kRight][2][2]], &aGray,
        &Q3CubeColors[cubieColors[kBack][0][2]], &aGray,
        &Q3CubeColors[cubieColors[kDown][2][2]]);

  }
  return myGroup;
}

void InitCube(
  CWindowPtr cubeWindow,
  const RGBColor cubeColors[6],
  const short cubieColors[6][3][3],
  short cubeWidth,
  short stepSize
) {
  long x, y, z;

  for (x = 0; x < 3; x++)
    for (y = 0; y < 3; y++)
      for (z = 0; z < 3; z++)
        gCubies[x][y][z] = 0;

  SetPort((GrafPtr)cubeWindow);

  gStepSize = stepSize;
  gTempCubeRotation = 0;
  gCubeRotation = 0;

  Q3Initialize();

  // sets up the 3d data for the scene
  // Create view for QuickDraw 3D.
  gView = MyNewView(cubeWindow, cubeWidth);

  // the main display group:
  gCubeModel = MyNewModel(cubeColors, cubieColors);

  // the drawing styles:
  gInterpolation =
      Q3InterpolationStyle_New(kQ3InterpolationStyleNone);
  gBackFacing = Q3BackfacingStyle_New(kQ3BackfacingStyleRemove);
  gFillStyle = Q3FillStyle_New(kQ3FillStyleFilled);

  DrawCube();
}

void QuarterTurn(
  CubeFace face,
  TurnDirection direction
) {
  long i, x, y, z;
  long aFirstX, aLastX, aFirstY, aLastY, aFirstZ, aLastZ;
  TQ3Matrix4x4  aCubieMatrix, aRotationMatrix;
  TQ3RotateAboutAxisTransformData aRotationdata;
  TQ3TransformObject aFaceRotation;
  TQ3GroupPositionaPos;
  TQ3GroupObject  aCubie;
  long stepsToTurn;

  aFirstX = 0;
  aLastX = 3;
  aFirstY = 0;
  aLastY = 3;
  aFirstZ = 0;
  aLastZ = 3;
```

Herman Miller Aeron Chair

www.sittingmachine.com

800-883-9697

```
// create a rotation object
switch(face)
{
  case kFront:
  {
    if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, -1.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, 1.0);
    aLastZ = 1;
    break;
  }
  case kBack:
  {
    if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, 1.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, -1.0);
    aFirstZ = 2;
    break;
  }
  case kLeft:
  {
    if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, 1.0, 0.0, 0.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, -1.0, 0.0, 0.0);
    aLastX = 1;
    break;
  }
  case kRight:
  {
if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, -1.0, 0.0, 0.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, 1.0, 0.0, 0.0);
    aFirstX = 2;
    break;
  }
  case kUp:
  {
    if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, -1.0, 0.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 1.0, 0.0);
    aLastY = 1;
    break;
  }
  case kDown:
  {
    if (direction == kClockwise)
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 1.0, 0.0);
    else
Q3Vector3D_Set(&aRotationdata.orientation, 0.0, -1.0, 0.0);
    aFirstY = 2;
    break;
  }
}
Q3Point3D_Set(&aRotationdata.origin, 0.0, 0.0, 0.0);
aRotationdata.radians = 0.0;

aFaceRotation = Q3RotateAboutAxisTransform_New(&aRotationdata);
// add the rotation object to each cubie in the face
for (x = aFirstX; x < aLastX; x++)
  for (y = aFirstY; y < aLastY; y++)
    for (z = aFirstZ; z < aLastZ; z++)
    {
      Q3Group_GetFirstPosition(gCubies[x][y][z], &aPos);
      Q3Group_AddObjectBefore(gCubies[x][y][z], aPos,
          aFaceRotation);
    }
// draw and adjust the angle
stepsToTurn = gStepSize / 4;
for (i = 1; i < stepsToTurn; i++)
{
  Q3RotateAboutAxisTransform_SetAngle(aFaceRotation,
        (2.0 * kQ3Pi * i / gStepSize));
  DrawCube();
}
```

```
// set the angle to 90° and adjust the rotation of each cubie
Q3RotateAboutAxisTransform_SetAngle(aFaceRotation,
        (kQ3Pi / 2.0));
Q3Transform_GetMatrix(aFaceRotation, &aRotationMatrix);
if (aFaceRotation)
  Q3Object_Dispose(aFaceRotation);
for (x = aFirstX; x < aLastX; x++)
  for (y = aFirstY; y < aLastY; y++)
    for (z = aFirstZ; z < aLastZ; z++)
    {
      Q3Group_GetFirstPosition(gCubies[x][y][z], &aPos);
    aFaceRotation = Q3Group_RemovePosition(gCubies[x][y][z],
          aPos);
      if (aFaceRotation)
        Q3Object_Dispose(aFaceRotation);
      Q3Group_GetFirstPositionOfType(gCubies[x][y][z],
          kQ3TransformTypeMatrix, &aPos);
      Q3Group_GetPositionObject(gCubies[x][y][z], aPos,
          &aFaceRotation);
      if (aFaceRotation)
      {
      Q3MatrixTransform_Get(aFaceRotation, &aCubieMatrix);
    Q3Matrix4x4_Multiply(&aCubieMatrix, &aRotationMatrix,
          &aCubieMatrix);
      Q3MatrixTransform_Set(aFaceRotation, &aCubieMatrix);
        Q3Object_Dispose(aFaceRotation);
      }
    }
DrawCube();

// rotate cubies in gCubies
switch(face)
{
  case kFront:
  {
    if (direction == kClockwise)
    {
      aCubie          = gCubies[0][0][0];
      gCubies[0][0][0] = gCubies[0][2][0];
      gCubies[0][2][0] = gCubies[2][2][0];
      gCubies[2][2][0] = gCubies[2][0][0];
      gCubies[2][0][0] = aCubie;
      aCubie          = gCubies[1][0][0];
      gCubies[1][0][0] = gCubies[0][1][0];
      gCubies[0][1][0] = gCubies[1][2][0];
      gCubies[1][2][0] = gCubies[2][1][0];
      gCubies[2][1][0] = aCubie;
    }
    else
    {
      aCubie          = gCubies[0][2][0];
      gCubies[0][2][0] = gCubies[0][0][0];
      gCubies[0][0][0] = gCubies[2][0][0];
      gCubies[2][0][0] = gCubies[2][2][0];
      gCubies[2][2][0] = aCubie;
      aCubie          = gCubies[1][2][0];
      gCubies[1][2][0] = gCubies[0][1][0];
      gCubies[0][1][0] = gCubies[1][0][0];
      gCubies[1][0][0] = gCubies[2][1][0];
      gCubies[2][1][0] = aCubie;
    }
    break;
  }
  case kBack:
  {
    if (direction == kClockwise)
    {
      aCubie          = gCubies[0][2][2];
      gCubies[0][2][2] = gCubies[0][0][2];
      gCubies[0][0][2] = gCubies[2][0][2];
      gCubies[2][0][2] = gCubies[2][2][2];
      gCubies[2][2][2] = aCubie;
      aCubie          = gCubies[1][2][2];
      gCubies[1][2][2] = gCubies[0][1][2];
      gCubies[0][1][2] = gCubies[1][0][2];
      gCubies[1][0][2] = gCubies[2][1][2];
      gCubies[2][1][2] = aCubie;
    }
```

```
    else
    {
      aCubie          = gCubies[0][0][2];
      gCubies[0][0][2] = gCubies[0][2][2];
      gCubies[0][2][2] = gCubies[2][2][2];
      gCubies[2][2][2] = gCubies[2][0][2];
      gCubies[2][0][2] = aCubie;
      aCubie          = gCubies[1][0][2];
      gCubies[1][0][2] = gCubies[0][1][2];
      gCubies[0][1][2] = gCubies[1][2][2];
      gCubies[1][2][2] = gCubies[2][1][2];
      gCubies[2][1][2] = aCubie;
    }
    break;
  }
  case kLeft:
  {
    if (direction == kClockwise)
    {
      aCubie          = gCubies[0][0][2];
      gCubies[0][0][2] = gCubies[0][2][2];
      gCubies[0][2][2] = gCubies[0][2][0];
      gCubies[0][2][0] = gCubies[0][0][0];
      gCubies[0][0][0] = aCubie;
      aCubie          = gCubies[0][0][1];
      gCubies[0][0][1] = gCubies[0][1][2];
      gCubies[0][1][2] = gCubies[0][2][1];
      gCubies[0][2][1] = gCubies[0][1][0];
      gCubies[0][1][0] = aCubie;
    }
    else
    {
      aCubie          = gCubies[0][2][2];
      gCubies[0][2][2] = gCubies[0][0][2];
      gCubies[0][0][2] = gCubies[0][0][0];
      gCubies[0][0][0] = gCubies[0][2][0];
      gCubies[0][2][0] = aCubie;
      aCubie          = gCubies[0][2][1];
      gCubies[0][2][1] = gCubies[0][1][2];
      gCubies[0][1][2] = gCubies[0][0][1];
      gCubies[0][0][1] = gCubies[0][1][0];
      gCubies[0][1][0] = aCubie;
    }
    break;
  }
  case kRight:
  {
    if (direction == kClockwise)
    {
      aCubie          = gCubies[2][2][2];
      gCubies[2][2][2] = gCubies[2][0][2];
      gCubies[2][0][2] = gCubies[2][0][0];
      gCubies[2][0][0] = gCubies[2][2][0];
      gCubies[2][2][0] = aCubie;
      aCubie          = gCubies[2][2][1];
      gCubies[2][2][1] = gCubies[2][1][2];
      gCubies[2][1][2] = gCubies[2][0][1];
      gCubies[2][0][1] = gCubies[2][1][0];
      gCubies[2][1][0] = aCubie;
    }
    else
    {
      aCubie          = gCubies[2][0][2];
      gCubies[2][0][2] = gCubies[2][2][2];
      gCubies[2][2][2] = gCubies[2][2][0];
      gCubies[2][2][0] = gCubies[2][0][0];
      gCubies[2][0][0] = aCubie;
      aCubie          = gCubies[2][0][1];
      gCubies[2][0][1] = gCubies[2][1][2];
      gCubies[2][1][2] = gCubies[2][2][1];
      gCubies[2][2][1] = gCubies[2][1][0];
      gCubies[2][1][0] = aCubie;
    }
    break;
  }
  case kUp:
  {
    if (direction == kClockwise)
```

```
      {
        aCubie          = gCubies[0][0][2];
        gCubies[0][0][2] = gCubies[0][0][0];
        gCubies[0][0][0] = gCubies[2][0][0];
        gCubies[2][0][0] = gCubies[2][0][2];
        gCubies[2][0][2] = aCubie;
        aCubie          = gCubies[1][0][2];
        gCubies[1][0][2] = gCubies[0][0][1];
        gCubies[0][0][1] = gCubies[1][0][0];
        gCubies[1][0][0] = gCubies[2][0][1];
        gCubies[2][0][1] = aCubie;
      }
      else
      {
        aCubie          = gCubies[2][0][2];
        gCubies[2][0][2] = gCubies[2][0][0];
        gCubies[2][0][0] = gCubies[0][0][0];
        gCubies[0][0][0] = gCubies[0][0][2];
        gCubies[0][0][2] = aCubie;
        aCubie          = gCubies[1][0][2];
        gCubies[1][0][2] = gCubies[2][0][1];
        gCubies[2][0][1] = gCubies[1][0][0];
        gCubies[1][0][0] = gCubies[0][0][1];
        gCubies[0][0][1] = aCubie;
      }
      break;
    }
    case kDown:
    {
      if (direction == kClockwise)
      {
        aCubie          = gCubies[2][2][2];
        gCubies[2][2][2] = gCubies[2][2][0];
        gCubies[2][2][0] = gCubies[0][2][0];
        gCubies[0][2][0] = gCubies[0][2][2];
        gCubies[0][2][2] = aCubie;
        aCubie          = gCubies[1][2][2];
        gCubies[1][2][2] = gCubies[2][2][1];
        gCubies[2][2][1] = gCubies[1][2][0];
        gCubies[1][2][0] = gCubies[0][2][1];
        gCubies[0][2][1] = aCubie;
      }
      else
      {
        aCubie          = gCubies[0][2][2];
        gCubies[0][2][2] = gCubies[0][2][0];
        gCubies[0][2][0] = gCubies[2][2][0];
        gCubies[2][2][0] = gCubies[2][2][2];
        gCubies[2][2][2] = aCubie;
        aCubie          = gCubies[1][2][2];
        gCubies[1][2][2] = gCubies[0][2][1];
        gCubies[0][2][1] = gCubies[1][2][0];
        gCubies[1][2][0] = gCubies[2][2][1];
        gCubies[2][2][1] = aCubie;
      }
      break;
    }
  }
}

void RotateCube(
  CubeAxis axis,
  TurnDirection direction,
  short stepsToTurn
) {
  TQ3RotateAboutAxisTransformData aRotationdata;
  TQ3Matrix4x4  aCubeRotationMatrix, aTempMatrix;
  long i;

  // create a rotation object
  switch (axis)
  {
    case kFrontBack:
    {
      if (direction == kClockwise)
  Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, -1.0);
      else
  Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 0.0, 1.0);
      break;
    }
    case kLeftRight:
    {
      if (direction == kClockwise)
  Q3Vector3D_Set(&aRotationdata.orientation, 1.0, 0.0, 0.0);
      else
  Q3Vector3D_Set(&aRotationdata.orientation, -1.0, 0.0, 0.0);
      break;
    }
    case kUpDown:
    {
      if (direction == kClockwise)
  Q3Vector3D_Set(&aRotationdata.orientation, 0.0, -1.0, 0.0);
      else
  Q3Vector3D_Set(&aRotationdata.orientation, 0.0, 1.0, 0.0);
      break;
    }
  }
  Q3Point3D_Set(&aRotationdata.origin, 0.0, 0.0, 0.0);
  aRotationdata.radians = 0.0;
  // the cube has been rotated, rotate the orientation of the rotation
  Q3MatrixTransform_Get(gCubeRotation, &aCubeRotationMatrix);
  Q3Vector3D_Transform(&aRotationdata.orientation,
      &aCubeRotationMatrix, &aRotationdata.orientation);
  gTempCubeRotation =
      Q3RotateAboutAxisTransform_New(&aRotationdata);
  // draw and adjust the angle
  for (i = 1; i < stepsToTurn; i++)
  {
    Q3RotateAboutAxisTransform_SetAngle(gTempCubeRotation,
        (2.0 * kQ3Pi * i / gStepSize));
    DrawCube();
  }
  // set the angle to 90° and adjust the rotation object of the cube
  Q3RotateAboutAxisTransform_SetAngle(gTempCubeRotation,
      (2.0 * kQ3Pi * stepsToTurn / gStepSize));
  Q3Transform_GetMatrix(gTempCubeRotation, &aTempMatrix);
  Q3MatrixTransform_Get(gCubeRotation, &aCubeRotationMatrix);
  Q3Matrix4x4_Multiply(&aCubeRotationMatrix, &aTempMatrix,
      &aCubeRotationMatrix);
  Q3MatrixTransform_Set(gCubeRotation, &aCubeRotationMatrix);
  // don't need gTempCubeRotation anymore, dispose it
  if (gTempCubeRotation)
    Q3Object_Dispose(gTempCubeRotation);
  gTempCubeRotation = 0;
  DrawCube();
}

void TermCube(void) {
  long x, y, z;

  Q3Object_Dispose(gView);
  Q3Object_Dispose(gCubeModel); // object in the scene being modelled
  Q3Object_Dispose(gCubeRotation);
  for (x = 0; x < 3; x++)
    for (y = 0; y < 3; y++)
      for (z = 0; z < 3; z++)
      {
        if (gCubies[x][y][z])
          Q3Object_Dispose(gCubies[x][y][z]);
  // object in the scene being modelled
      }
  Q3Object_Dispose(gInterpolation);  // interpolation style used when rendering
  Q3Object_Dispose(gBackFacing);
      // whether to draw shapes that face away from the camera
  Q3Object_Dispose(gFillStyle);
      // whether drawn as solid filled object or decomposed to components
  Q3Exit();
}
```


• Custom Destinations
• Electronic Transaction Processing
• Easy Trialware Creation
• Install or UnInstall
Get It Together With InstallerMaker™
• Installation From FTP or HTTP Sites
When Time Is Money, Get It Done Fast!
Build Smaller Installers – With StuffIt InstallerMaker, you'll build installers faster than ever before! Compress your installers an average of 15% smaller using the power of the StuffIt Engine®. Smaller installers download faster, provide added space on CDs and servers, and increase network bandwidth.
Quickly Create Demoware - Easily turn your application into a polished demo. Just set the number of days for the demo to be active, paste in the graphics, and you're done!
Archive Freshening - Automatically update your installer project file; eliminate repeated searches for modified files.
• Resource Installation
• Built-In Resource Compression
Scripting Saves Time - InstallerMaker provides full scriptability for all features. Automating the build process saves time and helps ensure the integrity of your installer.
• ShrinkWrap Disk Image Support
• Marketing Opportunities To Help You Make Money
Trialware in Minutes - Creating trialware is a breeze with InstallerMaker. With just a few clicks, and no extra coding, you can create trialware that is e-commerce ready.
Update in a Flash - Easily build intelligent "diff" files in installers to create small updaters for quick online distribution. From one simple installer, you can update 68k, PowerPC or FAT applications.
• Hierarchical Package Support
More Details Online - There's a lot more InstallerMaker can do for you. Get all the info at www.aladdinsys.com.
• Built-In Updaters
Aladdin Systems
© 2000 Aladdin Systems, Inc. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.

By Tom Djajadiningrat and Maarten Gribnau

Cubby: Multiscreen Desktop VR Part I

Multiple views and mirroring images in QuickDraw 3D

SUMMARY

In a series of three articles we describe how to implement the visualization part of a Cubby, a desktop virtual reality system which uses three orthogonally placed head-tracked screens. This series will give you an understanding of how this type of three dimensional display works and show you how easy it is to implement using off-the-shelf components. To facilitate implementation we use Apple's QuickDraw3D API. Even if you are not interested in virtual reality display technology, you may still be interested in the QuickDraw 3D techniques presented here. The most important ones are multiple views on a single model and the mirroring of images without the use of offscreen GWorlds.

INTRODUCTION

Cubby is a desktop virtual reality system developed at Delft University of Technology (Djajadiningrat et al, 1997; Djajadiningrat, 1998). Cubby uses three orthogonally placed head-tracked screens which form a cubic display space (**Figure 1**). Through the coupling of the perspectives on the screens to the head-movements of the user in real-time, the illusion is created that a virtual scene stands inside the display space. **Figure 2** shows a user in front of Cubby. **Figure 3** shows a chair inside Cubby's display space from four perspectives as generated by four different head positions. Because of the way the screens are placed, Cubby allows the virtual scene to be viewed from a wide range of visual angles (see movie 'visualization.mov'). And since the virtual scene appears in front of rather than behind the screens, the user can get at the objects in the virtual scene with an instrument without the screens forming an obstruction. This makes it possible to manipulate objects by means of an instrument at the place where they appear (see movie 'manipulation.mov'). The 3D impression that Cubby creates is based purely on head-tracking. It does not use stereo though of course this could be added.

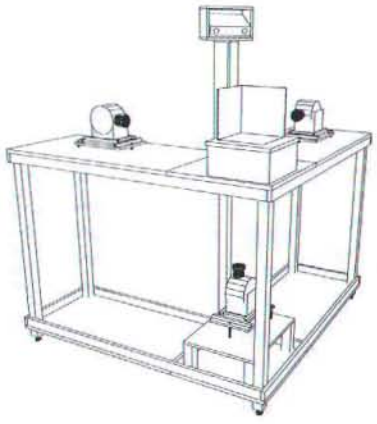


Figure 1. *The Cubby setup.*

Tom has calculated that if he could convince the current top contestants to refrain from taking part in the Programmer's Challenge and he were to submit a proposal for a Challenge every month, he could be leading the pack by as early as Christmas 2010. Just in case this clever ploy to achieve fame fails, he continues to hone his personal collection of irreconcilable skills.

Maarten lives in a binary world. His research is about two-handed interaction with 3D graphics. But recently, he realized that his bimanual interfaces fall short when his twins started to interact with his computer. He is now thinking about rewriting Nanosaur and implement four-handed interaction.

REAL PEOPLE, REAL SUPPORT.

MacTank provides technical support solutions for Macintosh application developers. We support your end users so you have time to concentrate on marketing and development. Bring your applications to OS X from Windows or NeXT and we will do the rest.

For pricing and information call 877-751-2300, option 2.

Visit us at www.mactank.com

MACTANK
THE TECH SUPPORT ALTERNATIVE

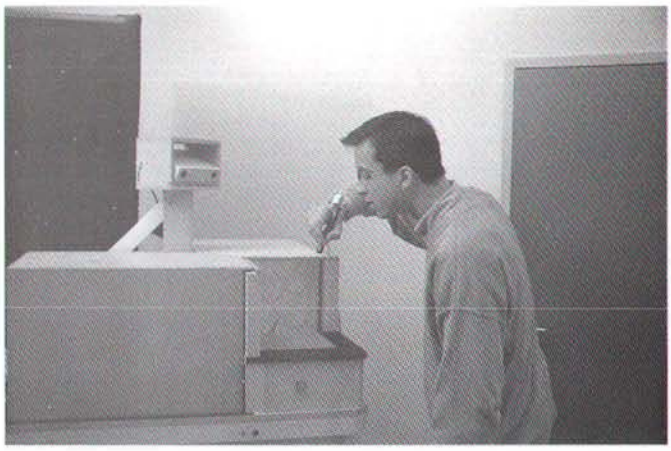


Figure 2. *A user in front of Cubby.*

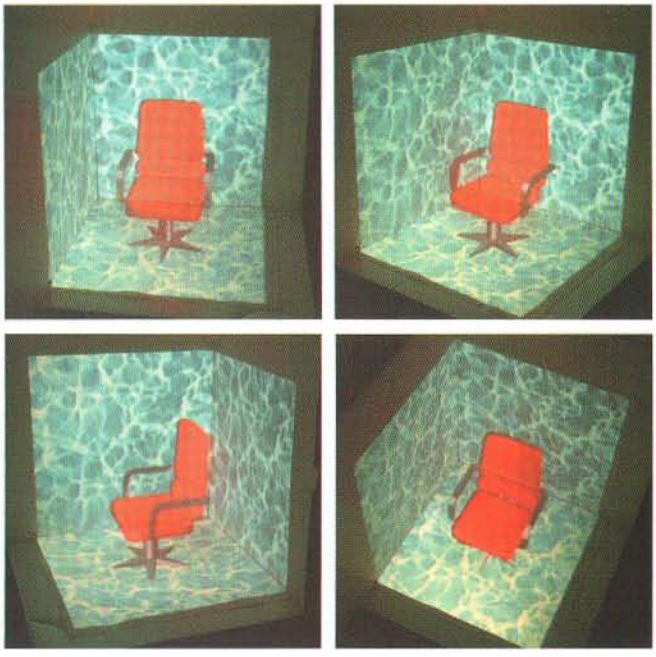


Figure 3. *Four perspectives on a desk chair as generated by different head positions.*

Technically, Cubby is similar to a CAVE (Cruz-Neira et al., 1993). A CAVE is a virtual reality environment in which the walls and floor of a room form projection screens. A CAVE measures approximately 3x3x3 metres, while Cubby's display space is only 0.2x0.2x0.2 metres. Thus from a technical point of view you can think of Cubby as a miniature CAVE. From an application point of view, however, Cubby and CAVE are quite different. With a CAVE, the user is inside the cubic space while with a Cubby the user is outside it. This gives each system its pros and cons. With a CAVE, the virtual scene is all around the user. It is therefore well-suited to panoramic viewing and walkthrough or rollercoaster types of simulation. Because of its large size it is difficult to realize accurate head and hand tracking with the currently available tracking technology. With a Cubby the user looks upon the virtual scene as if it were an object. Cubby's workspace is much smaller than CAVE's, but because of this, it is possible to realize accurate tracking of head-position and instruments. Cubby is therefore well-suited to precision tasks such as surgical simulation and computer aided design. Other advantages of Cubby are that it consumes little space, is relatively low-cost, and can be built using consumer grade off-the-shelf technology.

This article describes how to implement the visualization part of a Cubby in QuickDraw 3D. It will give you a grasp of the technology behind multiple screen head-tracked displays such as Cubby and CAVE. Even if you are not interested in such technology, you may still be interested in the QuickDraw 3D techniques presented here. They are multiple views on a single model and the mirroring of images without the use of offscreen GWorlds. We also include a section troubleshooting, to help you get Cubby running, and a section Tidbits, with suggestions to further improve Cubby.

Cubby on Power Macintosh

What you should know

We assume that you are familiar with the basics of QuickDraw 3D programming. If you have not dealt with QuickDraw 3D before we suggest that you have a look at the introduction to QuickDraw 3D in Develop 22 (Thompson and Fernicola, 1995) or at chapter nine 'QuickDraw 3D' of 'Tricks of the Mac Game Programming Gurus' (Greenstone, 1995).

We also assume that you are familiar with Part I and II of 'Desktop VR using QuickDraw 3D', two MacTech articles that appeared in July and August of 1998 (Djajadiningrat and Gribnau, 1998; Gribnau and Djajadiningrat, 1998).

Required hardware and software

To try out the QuickDraw 3D techniques in this article you need a PowerMacintosh with an accelerated 3D graphics board and QuickDraw 3D 1.5.4 or better.

If you wish to build an actual Cubby you need additional electronic hardware such as a head-tracker, three projectors and possibly extra graphics boards and scan converters. As a head-tracker we use a Dynasight infra-red tracker by Origin Instruments. With regard to projectors, graphics boards and scan converters, your exact needs depend on which configuration you choose. We will discuss possible configurations in a minute. Of course, a Cubby consists of more than electronics and computer hardware alone. You also need to build a physical setup to position

the projectors relative to the screens. For quick experimentation you can build the display space using cardboard and tracing paper or drafting foil and mount the projectors on tripods providing they are not too big. **Figure 4** shows what our first setup looked like. For a more permanent and robust setup you need to build the display space from 4-5mm thick projection material (available from professional photography labs) and mount the projectors on a table (**Figure 5**).

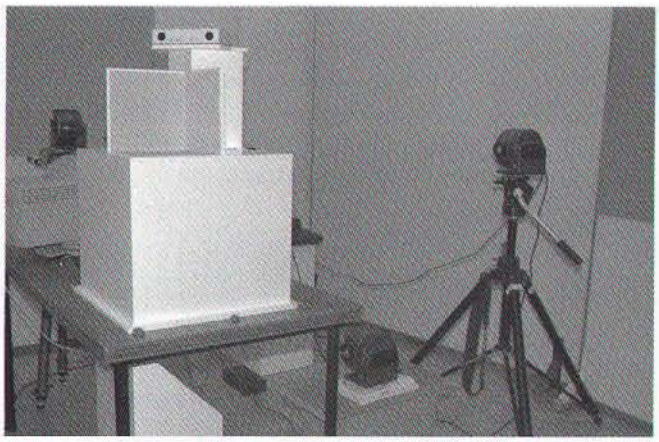


Figure 4. *Our preliminary setup with a display space built from foamboard and drafting foil. The projectors are mounted on tripods.*

Figure 5. *A more robust setup. This table makes it possible to accurately line up projectors and screens.*

Possible hardware configurations

Your Mac needs to generate three images, one per projection screen. You can these images to the three projectors in several ways. You can either work with one graphics board or three graphics boards and with either computer projectors or video projectors (A computer projector is a projector that can directly accept the VGA output of the graphics board of your Mac; a video projector is one which only accepts a composite or S-video signal such as produced by your home video recorder. A projector which accepts VGA usually accepts S-video too, but not all video projectors accept VGA input). This leads to four possible configurations (**Figure 6**).

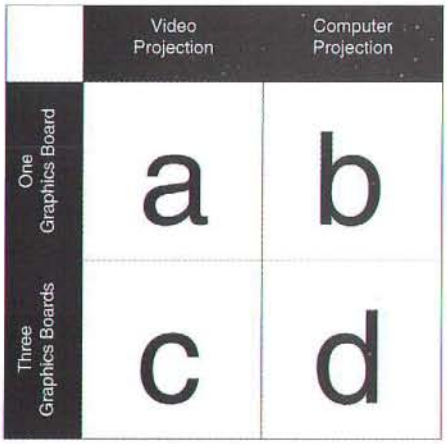


Figure 6. *A 2x2 matrix leading to four different configurations.*

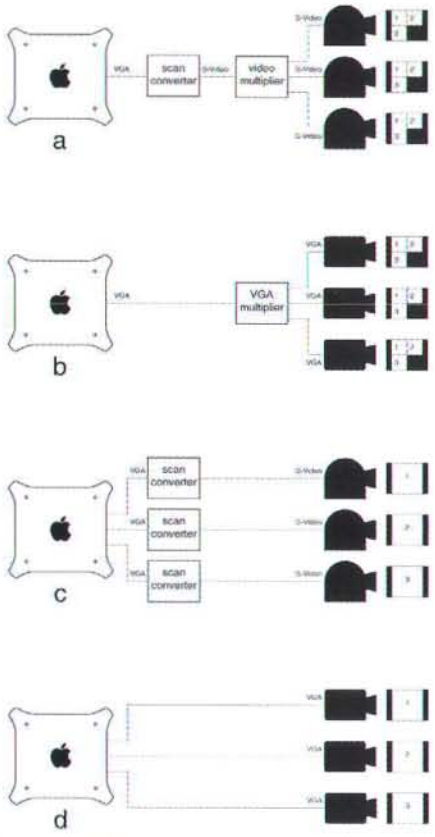


Figure 7. *The four different configuration a-d (top to bottom)*

The simplest configuration is to work with a single graphics board, use a multiplier to split the signal of this board into three identical signals, and feed each of these identical signals to a computer beamer (**Figure 7a**). Each projector is placed in such a way relative to its screen that the relevant part of the image appears on the screen, while the two remaining images are simply discarded by being projected off screen.

The second way is similar to the first, with the difference that we use a scan converter to convert the signal of the graphics board into a video signal, which is then split up by a video multiplier and fed to three video projectors (**Figure 7b**). While this results in lesser image quality it is likely to be less expensive. The extra costs for the scan converter are less than the cost you save by using consumer grade video projectors instead of computer beamers.

The disadvantage of using a single graphics board is that you do not make full use of the resolution of the projectors as approximately three quarters of the display area is discarded. A way to overcome this is to use three graphics boards instead of just one. This leads to a setup in which there is one graphics board per projector. Consequently, no splitter box is necessary. (**Figure 7c**). Of course, this assumes that your Mac has enough PCI slots free to add extra graphics boards.

The last configuration is similar to the third, the difference being that it uses video projectors rather than computer projectors (**Figure 7d**).

As always, there are clever ways to cut costs. For example, you can avoid the extra cost of one or more scan converters by using graphics boards which have both an output for a conventional monitor and a S-video output. Some desktop Macs and PowerBooks even have S-video outputs as part of their standard configuration.

A Look at the Cubby App

Before we dive into an explanation of the computer graphics behind Cubby, let's run the application to see what we are aiming for. Don't worry about connecting a head-tracker to the Mac, for the moment we run under mouse control. Start up the application 'Cubby' and open the model 'espresso.3df'. On screen you see an L-shape with an espresso pot (**Figure 8**). The L-shape comprises three QuickDraw 3D panes which are placed within a single window which covers the whole screen except for the menu bar (The advantage of using three panes within a single window instead of one window per view is that that the single window acts as a black backdrop to the panes. This prevents the Macintosh desktop from appearing at the outer edges of Cubby's screens). While the espressopot was loaded from disk, the background planes with the marbled texture form part of the Cubby application.

Now try moving the mouse. On the conventional 2D-monitor that you are using now, the three perspective views in the L-shape look strangely distorted. When the views are folded into a cube, as in Cubby's display space, and coupled

to the user's head-position the espresso pot appears to stand within Cubby's space.

Choose Mirrored from the Images menu. This mirrors each image horizontally (**Figure 9**). By looking at the frames per second counter in the statusbar and choosing Normal and Mirrored repeatedly you can see how mirroring does not impose much of a speed penalty.

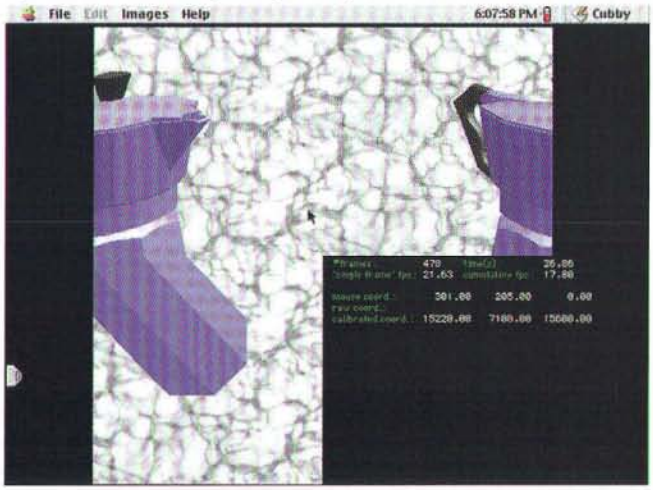


Figure 8. *A screen shot of the Cubby application (non-mirrored).*

Figure 9. *A screen shot of the Cubby application (mirrored).*

Multiple Head-Tracked Displays

The easiest way to think of Cubby is as three head-tracked displays. Each of the back-projection screens is a head-tracked display. For an extensive discussion of head-tracked displays please refer to MacTech July 1998 (Djajadiningrat & Gribnau, 1998). In that issue we explained that for a head-tracked display an off-axis rather than an on-axis camera is needed. In QuickDraw 3D speak this means a view plane camera rather than a aspect ratio camera. For Cubby we use three QuickDraw 3D views each of which has its own view plane camera. Each of the **Figures 10** to **12** shows one view plane camera, its coordinates, its viewing

pyramid and its line of sight. The point at which Cubby's three screens meet is placed at the origin of the world coordinate system. Notice how the three view plane cameras share the same location C with coordinates Cx, Cy and Cz. Later we will couple the camera's location to the head-position of the user. While the three cameras share the same location, each one has a different orientation as the line of sight of each camera is at right angles to its screen. The orientation of a view plane camera is determined by its location and the point of interest, point P in the figures. **Figure 10** shows the camera for the screen in the Z=0 plane with the line of sight parallel to the z-axis. **Figure 11** shows the camera for the screen in the X=0 plane with the line of sight parallel to the x-axis. Finally, **Figure 12** shows the camera for the screen in the Y=0 plane with its line of sight parallel to the vertical y-axis.

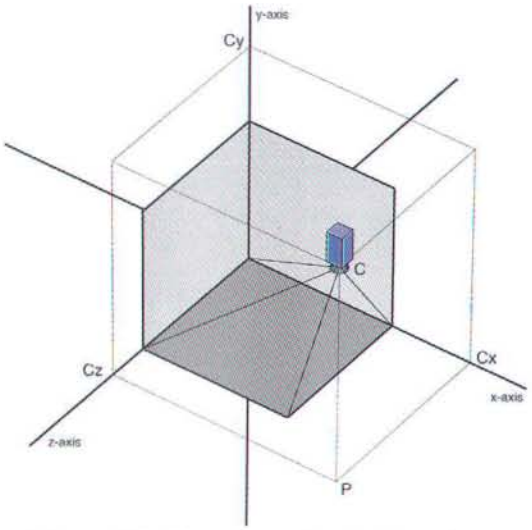


Figure 10. *The camera for the Z=0 plane.*

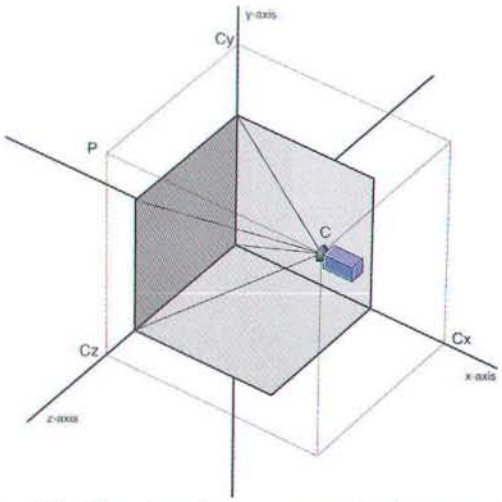


Figure 11. *The view plane camera for the X=0 plane.*

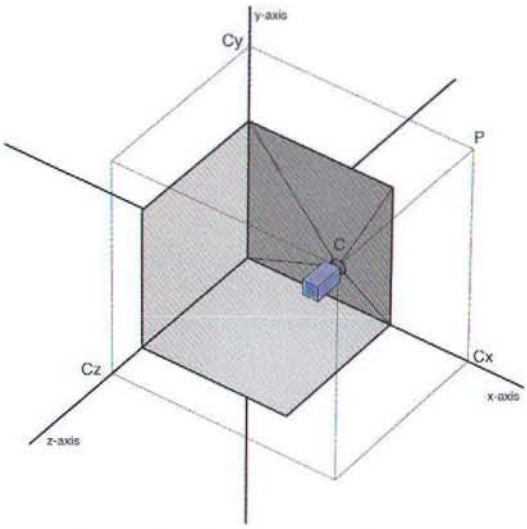


Figure 12. *The view plane camera for the Y=0 plane.*

Mirroring without GWorlds

To get a convincing depth impression in Cubby the three screens need to form a single, seamless display. Therefore we cannot use conventional monitors or flat-panel displays, which always have a frame around their imageable area which would result in a disturbing seam (**Figure 13**). Instead we have to use projection screens. Since we are projecting on the back of the screens, we need to mirror the images that are sent to the projectors. So, how do we do this? Some projectors allow mirroring in hardware. But many projectors, especially the low-end ones, lack this feature which means that you have to mirror the images in software. The conventional approach to software mirroring would go like this. First, we would render an image to an offscreen GWorld. Then we would need to copy this image to a second offscreen GWorld so that the image is mirrored. This could be done by copying the n-th one pixel-wide column of the first GWorld to the (width-n)th column of the second GWorld. Finally, we would copy this second GWorld to the screen. There are two problems with this approach. The first is that many graphics boards do not support hardware acceleration when rendering to offscreen GWorlds with QuickDraw 3D which would make rendering slow. The second is that the copying between the GWorlds takes too much time. And to get a convincing depth impression with a head-tracked display we need all the speed we can get.

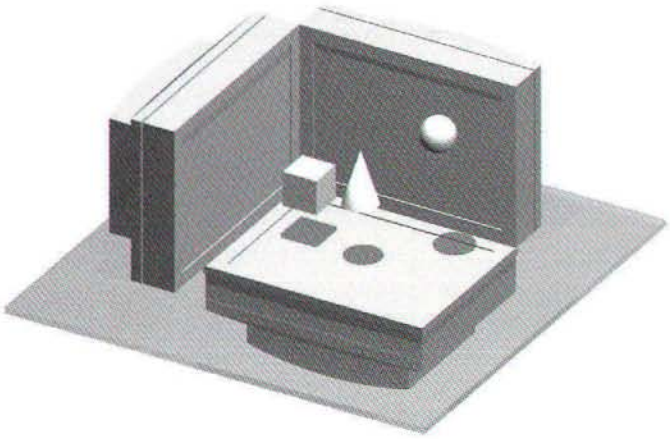


Figure 13. *We cannot use ordinary monitors to build Cubby's display space because of the borders around the imageable areas.*

Luckily, there is another way to mirror the images which does not require offscreen GWorlds. This involves mirroring the virtual scene, the lights and each camera to a different octant and reversing the orientation style of the scene. Remember that three-dimensional space is divided into eight octants by the three perpendicular coordinate planes. Figures 14 to 16 show the mirroring of the camera and the background planes (the mirroring of the model and the lights is not shown). Each camera is mirrored in the plane for which it is intended. Let's consider each camera in turn. The original camera position is C, the mirrored position C'. Figure 14 shows how the camera for the Z=0 plane is mirrored in the Z=0 plane. **Figure 15** shows how the camera for the X=0 plane is mirrored in the X=0 plane. Finally, Figure 16 shows how the camera for the Y=0 plane is mirrored in the Y=0 plane. Enough talk, let's look at the code. We split it up into two parts: the code concerning multiple views and the code for mirroring.

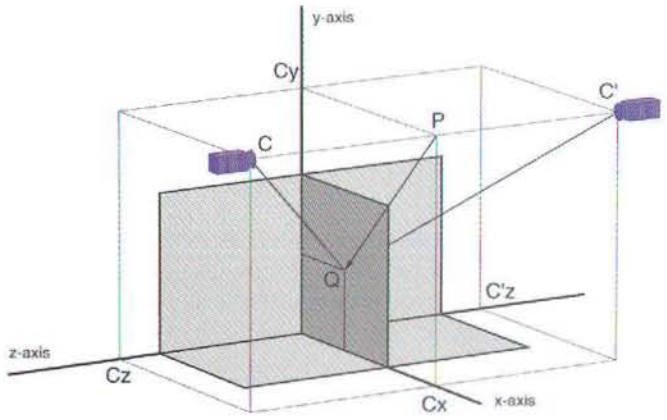


Figure 14. *Mirroring in the Z=0 plane.*

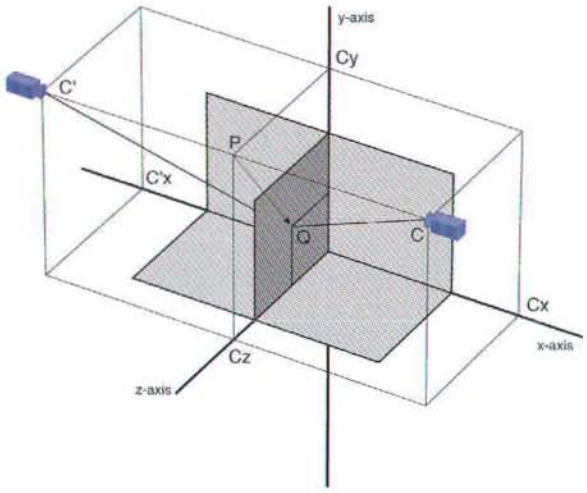


Figure 15. *Mirroring in the X=0 plane.*

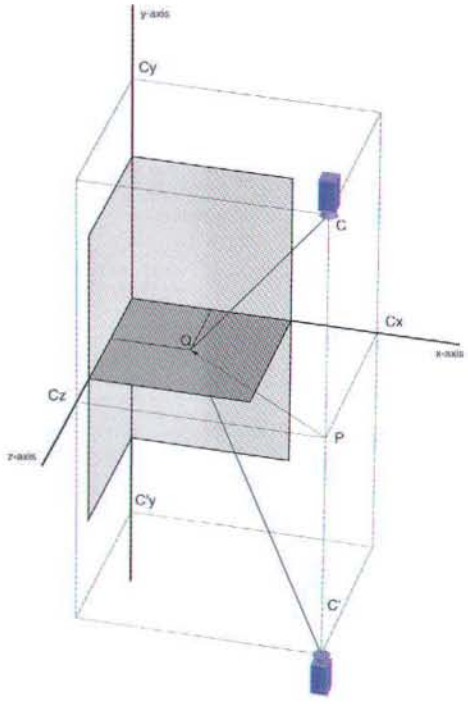


Figure 16. *Mirroring in the Y=0 plane.*

Multiple Views

In this section we discuss:

- creating multiple views
- adjusting the view plane cameras
- submitting the views for rendering

Creating multiple views

So, how do we go about creating the three views? We use a global struct gDoc of type DocumentRec with the fields fView1, fView 2 and fView3 which contain pointers to the three views of Cubby. The DocumentRec struct is defined in Shell.h. You will see that we use this struct a lot. It is a simple way of passing around often used variables such as view objects and matrices.

From Init in Shell.c (Listing 1) we call initDocumentData in (De)Init.c. The global struct gDoc is passed as a parameter.

Listing 1: Shell.c

Init

```
InitDocumentData(&gDoc, gWindow);
```

In InitDocumentData we fill in the fields fView1, fView2 and fView3 of the ioDoc parameter by calling newView in ViewCreation.c three times and passing the rectangles of the panes within our window (Listing 2).

Listing 2: (De)Init.c

InitDocumentData(DocumentPtr ioDoc, WindowPtr inWindow)

```
Rect theRect1=(kPane1T, kPane1L, kPane1B, kPane1R);
Rect theRect2=(kPane2T, kPane2L, kPane2B, kPane2R);
Rect theRect3=(kPane3T, kPane3L, kPane3B, kPane3R);

ioDoc->fView1=newView(ioDoc, inWindow, &theRect1);
ioDoc->fView2=newView(ioDoc, inWindow, &theRect2);
ioDoc->fView3=newView(ioDoc, inWindow, &theRect3);
```

The routine newView should need no further explanation as it forms a standard part of each QuickDraw 3D application. It calls newDrawContext and passes in a rectangle to set up a pane within a window (Listing 3).

Listing 3: ViewCreation.c

newDrawContext (WindowPtr inWindow, Rect *inRect)

```
theDrawContextData.paneState    = kQ3True;
theDrawContextData.pane.min.x = inRect->left;
theDrawContextData.pane.min.y = inRect->top;
theDrawContextData.pane.max.x = inRect->right;
theDrawContextData.pane.max.y = inRect->bottom;
```

The routine newDrawContext returns a new draw context to newView. It then goes on to create and set a renderer, a view plane camera and a light group. As you can see, creating multiple views is quite simple. It is a matter of repeating the view object creation as found in each basic QuickDraw 3D application.

Adjusting the three view plane cameras

Now we come to the most interesting part of the code: adjusting the view plane camera of each view to the current position of the head-tracker (Listing 4). We need to convert the raw head-position, expressed in the head-trackers own coordinate system, to the world coordinate system. This is done by calling Q3Point3D_Transform with the calibration matrix in the fCalMatrix field of our struct gDoc which was passed as a parameter to AdjustCameras. We will discuss how to create this matrix in the third part of this series.

Once we have a camera position in world coordinates, we need to ascertain that it stays within the positive XYZ octant. Just before the camera threatens to move outside this octant, the camera's hither plane is pushed through the

background plane causing the latter to disappear. This is a disturbing effect which we can avoid by carefully tuning the minimum allowed distance between camera and background plane to the hither value of the camera. Here we set the minimum allowed distance between camera and background plane to kHither+kMargin QuickDraw 3D units. A user of Cubby who comes closer to the screens than the real world equivalent of kHither+kMargin QuickDraw 3D units will notice that the perspectives do not completely follow his head movements, but at least the background plane on that screen does not completely disappear. In the section Tidbits we discuss a more elegant way to solve this problem.

AdjustCameras finishes by calling AdjustOneCamera for each view and passing the camera position as a parameter.

Listing 4: ViewPlaneCamera.c

AdjustCameras

```
TQ3Point3D   H, C;

// apply the calibration matrix to convert the
// raw head position to a camera position
// in world coordinates.

Q3Point3D_Transform(&H,
          &inDoc->fCalMatrix,
          &C);

// Limit the camera position so
// we do not move beyond the screens.
if (C.x<= kHither) C.x= kHither ;
if (C.y<= kHither) C.y= kHither ;
if (C.z<= kHither) C.z= kHither ;

// Adjust the camera of each view.
AdjustOneCamera(inDoc, &C, kView1);
AdjustOneCamera(inDoc, &C, kView2);
AdjustOneCamera(inDoc, &C, kView3);

return kQ3Success ;

bail:
return kQ3Failure ;
```

Let's look at AdjustOneCamera (Listing 6). In this routine we adjust a view plane camera of a single view by getting the camera object from the view and updating its parameters. Our objective is to be able to fill in the TQ3ViewPlaneCameraData struct as defined in QD3DCamera.h (Listing 5). This means that we also need to be able to fill in a TQ3CameraData struct, a TQ3CameraPlacement struct and a TQ3CameraRange struct. Luckily, some variables do not change as we adjust our cameras. The halfWidthAtViewPlane, halfHeightAtViewPlane and viewPort variables stay the same during the execution of our program.

Listing 5

```
struct TQ3ViewPlaneCameraData {
  TQ3CameraData       cameraData;
  float               viewPlane;
  float               halfWidthAtViewPlane;
  float               halfHeightAtViewPlane;
  float               centerXOnViewPlane;
  float               centerYOnViewPlane;
};

struct TQ3CameraData {
  TQ3CameraPlacement placement;
  TQ3CameraRange     range;
  TQ3CameraViewPort  viewPort;
};

struct TQ3CameraPlacement {
  TQ3Point3D          cameraLocation;       TQ3Point3D
      pointOfInterest;
  TQ3Vector3D         upVector;
};

struct TQ3CameraRange {
  float               hither;
  float               yon;
};
```

We start by setting the camera location C to a local copy of the camera location parameter inC that was passed to AdjustOneCamera. We also set the point of interest to the inC parameter (Notice that the term point of interest is somewhat misleading. The point of interest is the direction in which the camera is pointing but with a view plane camera it need not be visible as part of our image).

The next thing is to determine which view we are dealing with. We do this through a switch statement which looks at the inViewNumber parameter. In each case of the switch statement we get the camera object from the view, adjust the point of interest P, and set the distance to the view plane. We discuss the cases one by one.

If the inViewNumber equals kView1, we are dealing with the view on the Z=0 plane (**Figure 14**). First we get the camera object from the view by calling Q3View_GetCamera. As the point

of interest for the camera of this view is the projection of the camera location on the Z=0 plane, we set P.z to zero. To describe the settings of a view plane camera we also need the distance from the camera to the view plane, held in theViewPlane variable. The camera for kView1 looks parallel to the Z-axis so the distance equals the z-coordinate of the camera location.

If the inViewNumber equals kView2, we are dealing with the view on the X=0 plane (**Figure 15**). Again we start by getting the camera object from the view by calling Q3View_GetCamera. As the point of interest for the camera of this view is the projection of the camera location on the X=0 plane, we set P.x to zero. The camera for kView2 looks parallel to the X-axis so the distance from the camera to the view plane equals the x-coordinate of the camera location.

Finally, if the inViewNumber equals kView3, we are dealing with the view on the Y=0 plane (**Figure 16**). Again we start by getting the camera object from the view by calling Q3View_GetCamera. As the point of interest for the camera of this view is the projection of the camera location on the Y=0 plane, we set P.y to zero. The camera for kView3 looks parallel to the Y-axis so the distance from the camera to the view plane equals the y-coordinate of the camera location. With this view we need to take care of one more thing. As the camera is looking down on the Y=0 plane as if it were a photographic enlarger, we need to change the up vector of the camera from (0,1,0) to (0,0,-1). The up vector for this view's camera points in the direction of the negative z-axis.

Now we need to look at the centerXOnViewPlane and the centerYOnViewPlane settings of the view plane camera. These determine the centre of the part of the view plane that we are interested in, indicated by an Q in **Figures 14** to **16**. In world coordinates point Q does not change, but as centerXOnViewPlane and centerYOnViewPlane are expressed in the camera's coordinate system they need to be adjusted if the camera is moved (For the moment we only consider non-mirrored cameras, later we will discuss what happens when we mirror the cameras. So we ignore the if (!gMirrored) statements for the moment and return to those later.).

CenterXOnViewPlane is the x-coordinate of the projection of the vector PY on the view plane in terms of camera's coordinate system. Likewise, CenterYOnViewPlane is the y-coordinate of the projection of the vector PY on the view plane in terms of camera's coordinate system. Let's look at CenterXOnViewPlane and CenterYOnViewPlane for each of the three views.

Figure 14 shows that for kView1, looking at the plane Z=0:

```
theCenterX = -C.x + kHalfWidthAtViewPlane;
theCenterY = -C.y + kHalfWidthAtViewPlane;
```

In Listing 6 the code says:

```
theCenterX = -C.x + kHalfWidthAtViewPlane + kFO;
theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;
```

So what is this mysterious kFO? Well, this is a Fiddle Offset constant. If you look in the header MyDefines.h you will see that it equals 0.001. Set it to 0, compile and run the Cubby app. You will see that suddenly the backgrounds start to flicker. Obviously, QuickDraw 3D is not very happy if the point of interest is a perfect projection on the view plane and it has to render polygons which lie exactly in the view plane. This is why we added the Fiddle Offset. kFO is so small that it does not cause any distortion but it does get rid of the flickering.

Figure 15 shows that for kView2, looking at the plane X=0:

```
theCenterX = +C.z - kHalfWidthAtViewPlane + kFO;
theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;
```

Finally, **Figure 16** tells us that for kView3, looking at the plane Y=0:

```
theCenterX = -C.x + kHalfWidthAtViewPlane + kFO;
theCenterY = +C.z - kHalfWidthAtViewPlane + kFO;
```

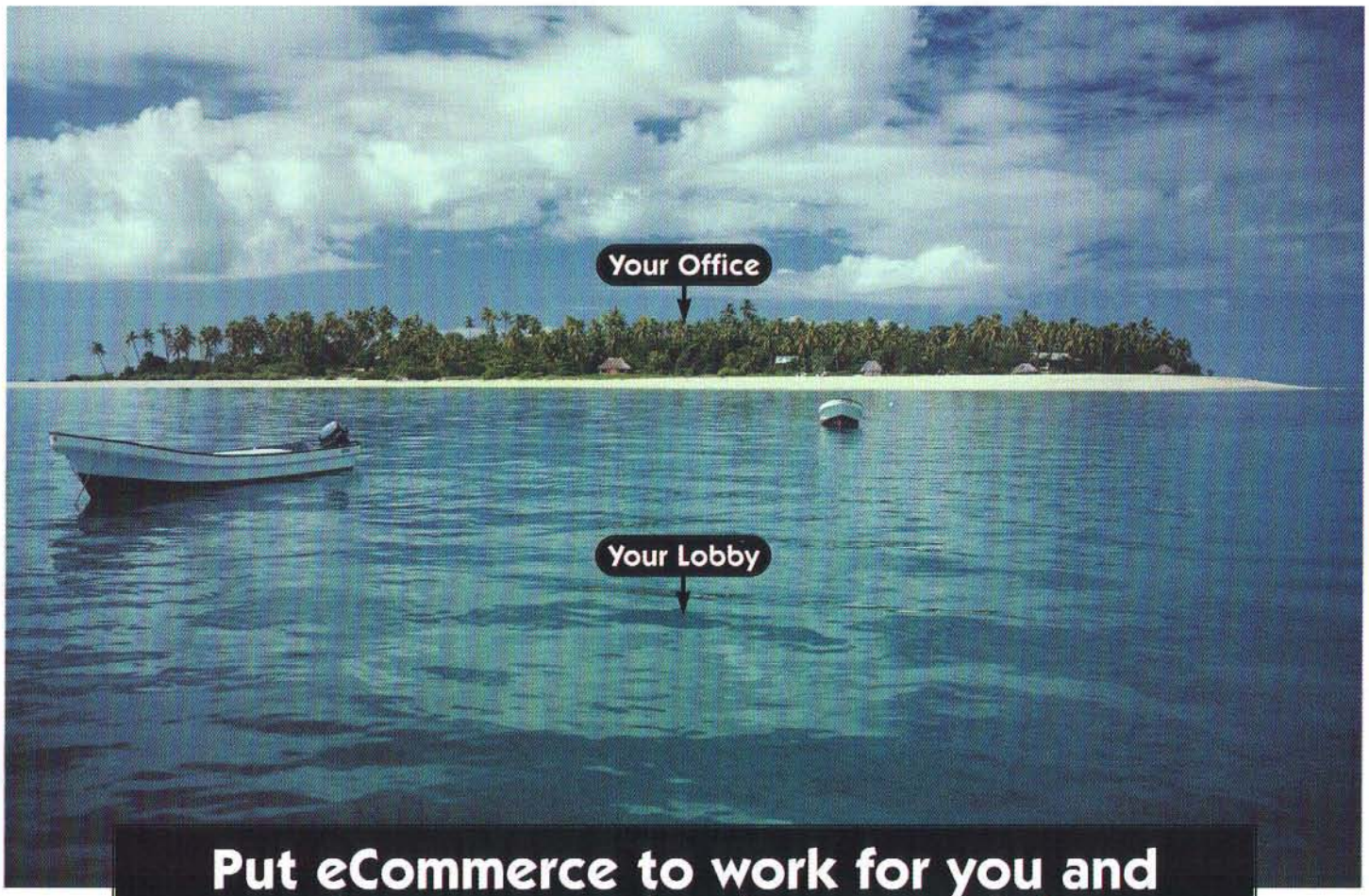
The last variables that we have to adjust are the hither and yon planes. Remember that we set a minimum distance of kHither+kMargin QuickDraw 3D units for the camera to approach the X=0, Y=0 and Z=0 planes. Here you see what the kMargin constant is good for: we avoid pushing the hither plane through the background planes by a margin of kMargin. A background plane is not only clipped if it lies between the camera location and the hither plane, but also if it lies beyond the yon plane. We prevent this from happening by setting the yon value to the view plane distance plus kMargin QuickDraw 3D units.

We complete the routine AdjustOneCamera by disposing the camera object through Q3Object_Dispose to balance the reference count.

Listing 6: ViewPlaneCamera.c

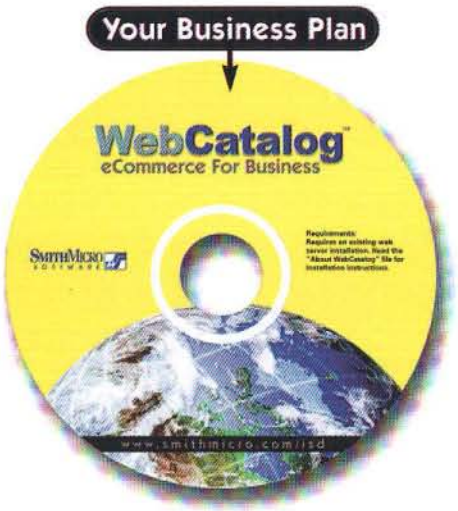
AdjustOneCamera

```
void AdjustOneCamera(DocumentPtr inDoc,
                     TQ3Point3D *inC,
                     short inViewNumber)
{
  TQ3CameraObject   theCamera ;
  TQ3CameraPlacement  theCameraPlacement;
  TQ3Point3D          C, P;
  TQ3Vector3D         theUp = { 0.0, 1.0, 0.0 };
  float               theViewPlane;
  float               theCenterX;
  float               theCenterY;
  TQ3CameraRange      theRange;

  // Make a local copy of the camera location.
  Q3Point3D_Set(&C, inC->x, inC->y, inC->z);

  // Determine the point of interest.
  // It is derived from the camera location,
  // so start with a copy of the camera location.
  Q3Point3D_Set(&P, inC->x, inC->y, inC->z);

  switch (inViewNumber)
  {
    case kView1:

      // Get the camera from the view
      Q3View_GetCamera(inDoc->fView1, &theCamera);

      P.z = 0.0;
      theViewPlane = C.z;

      break;
```



Put eCommerce to work for you and change your business model.

WebCatalog is a complete eCommerce and dynamic web publishing solution. Utilizing existing web server software, WebCatalog provides the capability to design and operate online storefronts with all the features found in leading sites. Build unlimited storefronts with our Storebuilder wizard directly from your browser - even upload your product graphics! WebCatalog provides multiple platform support for Windows, Macintosh and UNIX. It's the easiest way to take advantage of the web and to change how...and where... you do business.



- **Unlimited storefronts**
- **High-speed internal database**
- **WAP-enabled sites**
- **ODBC, SQL support**
- **Electronic shopping cart**
- **Invoice calculations**

1.800.477.1543 • www.smithmicro.com/isd

SMITHMICRO SOFTWARE

```
    case kView2:

      // Get the camera from the view
      Q3View_GetCamera(inDoc->fView2, &theCamera);

      P.x = 0.0;
      theViewPlane = C.x;

      break;

    case kView3:

      // Get the camera from the view
      Q3View_GetCamera(inDoc->fView3, &theCamera);

      P.y = 0.0;
      theViewPlane = C.y;

      // As this camera is pointing parallel to
      // the Y-axis, we need to change the theUp vector.
      Q3Vector3D_Set(&theUp, 0, 0, -1);

      break;
  }

  if (!gMirrored)
  {
    switch (inViewNumber)
    {
      case kView1:
        theCenterX = -C.x + kHalfWidthAtViewPlane + kFO;
        theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;

        break;

      case kView2:
        theCenterX = +C.z - kHalfWidthAtViewPlane + kFO;
        theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;

        break;

      case kView3:
        theCenterX = -C.x + kHalfWidthAtViewPlane + kFO;
        theCenterY = +C.z - kHalfWidthAtViewPlane + kFO;

        break;
    }
  }
  else
  {
    switch (inViewNumber)
    {
      case kView1:
        // Mirror in Z=0 plane.
        C.z = -C.z;
        theCenterX = +C.x - kHalfWidthAtViewPlane + kFO;

        theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;
        break;

      case kView2:
        // Mirror in X=0 plane.
        C.x = -C.x;
        theCenterX = -C.z + kHalfWidthAtViewPlane + kFO;
        theCenterY = -C.y + kHalfWidthAtViewPlane + kFO;
        break;

      case kView3:
        // Mirror in Y=0 plane.
        C.y = -C.y;
        theCenterX = +C.x - kHalfWidthAtViewPlane + kFO;
        theCenterY = +C.z - kHalfWidthAtViewPlane + kFO;
        break;
    }
  }

  // Adjust the hither and yon planes.
  theRange.hither= kHither;
  theRange.yon    = theViewPlane + kMargin;

  // Fill in the camera placement.
  theCameraPlacement.cameraLocation  = C;
  theCameraPlacement.pointOfInterest   = P;
  theCameraPlacement.upVector          = theUp;

  // Fill in the fields of the camera
  Q3Camera_SetPlacement(theCamera, &theCameraPlacement);
  Q3ViewPlaneCamera_SetViewPlane (theCamera, theViewPlane);
  Q3ViewPlaneCamera_SetCenterX     (theCamera, theCenterX);
  Q3ViewPlaneCamera_SetCenterY     (theCamera, theCenterY);
  Q3Camera_SetRange(theCamera, &theRange);

  // Dispose of the camera object
  Q3Object_Dispose( theCamera ) ;
}
```

Submitting three views for rendering

Now that we have adjusted the view plane camera of all three views, we can submit the views for rendering. Have a look at Listing 7 which shows the routine DocumentDraw3DData. The routine starts by calling Q3View_Sync for all three views. This is to force all three views to finish rendering the previous frame before we start rendering the next one. Again we ignore the if (gMirrored) statements for the moment. The remaining part of DocumentDraw3Ddata calls SubmitOneView in a rendering loop for each of the three views.

Listing 7: Rendering.c

DocumentDraw3DData

```
TQ3Status SubmitViews( DocumentPtr inDoc )
{
  TQ3Status  theStatus ;

  Q3View_Sync(inDoc->fView1);
  Q3View_Sync(inDoc->fView2);
  Q3View_Sync(inDoc->fView3);


  if (gMirrored)
    inDoc-> fMirrorMatrix = inDoc->fMatrixMirrorZ0;


  // The rendering loop for fView1
  Q3View_StartRendering( inDoc->fView1 );
  do
  {
    theStatus = SubmitOneView(inDoc, &(inDoc->fView1));
  }
  while ( Q3View_EndRendering(inDoc->fView1)
      == kQ3ViewStatusRetraverse );


  if (gMirrored)
    inDoc-> fMirrorMatrix = inDoc->fMatrixMirrorX0;


  // The rendering loop for fView2
  Q3View_StartRendering( inDoc->fView2 );
  do
  {
    theStatus = SubmitOneView(inDoc, &(inDoc->fView2));
  }
  while ( Q3View_EndRendering(inDoc->fView2)
      == kQ3ViewStatusRetraverse );

  if (gMirrored)
    inDoc-> fMirrorMatrix = inDoc->fMatrixMirrorY0;


  // The rendering loop for fView3
  Q3View_StartRendering( inDoc->fView3 );
  do
```

```
    theStatus = SubmitOneView(inDoc, &(inDoc->fView3));
  }
  while ( Q3View_EndRendering(inDoc->fView3)
      == kQ3ViewStatusRetraverse );

  return theStatus ;
}
```

So we turn our attention to SubmitOneView (Listing 8) which, you guessed it, submits a single view for rendering. Again we ignore the if(gMirrored) statement for the moment. The most important aspect of this routine is that we first submit the display group fDisplaySpace which contains the background planes, then the shader object and finally the model which was loaded from disk. If we would submit the background planes after the shading object the lighting on the background planes would change with the user's head-position. By submitting the background planes first we avoid getting such disturbing shading effects. The model which was loaded from disk is shaded as normal as it is submitted after the shading object.

Listing 8: Rendering.c

SubmitOneView

```
TQ3Status SubmitOneView(  DocumentPtr inDoc,
                          TQ3ViewObject inView)
{
  // if we wish to mirror the image, we need
  // to submit a mirroring matrix and change
  // the orientation style from counter clockwise
  // to clockwise.
  if (gMirrored)
  {
    Q3MatrixTransform_Submit( &inDoc-> fMirrorMatrix,
                              inView) ;
    Q3Style_Submit( inDoc->fOrientationStyle,
                    inView);
  }

  Q3Style_Submit(inDoc->fInterpolation,    inView);
  Q3Style_Submit(inDoc->fBackFacing,     inView);
  Q3Style_Submit(inDoc->fFillStyle,      inView);

  // submit the background planes which form
  // the cubic display space before we submit
  // the shader. That way we get evenly lit
  // background planes, regardless of where the
  // cameras are.
  Q3DisplayGroup_Submit( inDoc->fDisplaySpace, inView);

  // submit shader and styles
  Q3Shader_Submit(inDoc->fIlluminationShader, inView);

  // fit the model to the cubic display space
  Q3MatrixTransform_Submit(&inDoc->fModelMatrix, inView);

  // submit the model which was loaded from disk
  Q3DisplayGroup_Submit( inDoc->fModel, inView);

  return kQ3Success ;
}
```

Mirroring the Views

If your projectors do not feature hardware mirroring, you have to mirror the images in software. Perfect mirroring not only requires mirroring the camera and the virtual scene (including the background planes), but also mirroring the

lights and changing the orientation style. The order in which we discuss things is:

- creating the mirroring matrices
- setting the mirroring global
- mirroring the lights
- mirroring the three view plane cameras
- mirroring the virtual scene
- adjusting the orientation style

Creating the mirroring matrices

As we saw at the beginning of this article we need to do mirroring in the X=0, Y=0 and Z=0 planes. Listing 9 shows a code snippet for creating the three matrices that we need for these mirroring operations.

Listing 9 (De)Init.c

InitDocumentData

```
Q3Matrix4x4_SetScale(&ioDoc->fMatrixMirrorX0, -1,  1,  1);
Q3Matrix4x4_SetScale(&ioDoc->fMatrixMirrorY0,  1, -1,  1);
Q3Matrix4x4_SetScale(&ioDoc->fMatrixMirrorZ0,  1,  1, -1);
```

We use the QuickDraw 3D routine Q3Matrix4x4_SetScale to create each matrix. Scaling by –1 along one axis results in mirroring in the perpendicular plane. For example, scaling by –1 in along the x-axis results in mirroring in the plane X=0.

Setting the mirroring global

Next we look what happens when the user chooses Normal or Mirrored from the Images menu. Menu interaction is handled by the routine DoMenuCommand in Shell.c. Listing 10 shows an excerpt for the images menu. First HandleMenuCheckedItem is called which handles the checkmark in front of the menu items. Then the boolean gMirrored is toggled which determines whether to mirror the images or not. Finally, we call the routine mirrorLights for each view and pass the appropriate mirroring matrix as a parameter.

Listing 10 Shell.c

DoMenuCommand

```
case mImages:
switch (item)
{
  case iNormal:

    if (gMirrored == true)
    {
      HandleMenuCheckedItem(item);
      gMirrored = false;

      mirrorLights(gDoc.fView1, gDoc.fMatrixMirrorZ0);
      mirrorLights(gDoc.fView2, gDoc.fMatrixMirrorX0);
      mirrorLights(gDoc.fView3, gDoc.fMatrixMirrorY0);
    }
    break;
  case iMirrored:
    if (gMirrored == false)
    {
      HandleMenuCheckedItem(item);
      gMirrored = true;
      mirrorLights(gDoc.fView1, gDoc.fMatrixMirrorZ0);
      mirrorLights(gDoc.fView2, gDoc.fMatrixMirrorX0);
      mirrorLights(gDoc.fView3, gDoc.fMatrixMirrorY0);
    }
    break;
}
```

Mirroring the lights

We need to make sure that we mirror all the lights that belong to a view. When we created the view (newView in ViewCreation.c) we put all the lights in a light group object. What we have to do now is to obtain the light group from the view, traverse the light group, determine the type of each light and mirror the aspects relevant to that light. This is detailed in Listing 11.

We use the QuickDraw 3D routine Q3Light_GetType to determine the type of light. Through a switch statement we get cases for all possible types of light: ambient lights, point lights, directional lights and spot lights. An ambient light needs no mirroring as it has neither a location nor a direction. A point light has a location but no direction so we only need to mirror the former. A directional light can be thought of as the opposite of a point light: it has no location but it does have a direction so we only need to mirror the latter. Finally, there are spot lights which have both a location and a direction, so we need to mirror both.

Listing 11 MirrorLights.c

mirrorLights

```
TQ3Status mirrorLights(TQ3ViewObject inView,
                       TQ3Matrix4x4 inMatrix)
{
  TQ3GroupObject    theGroup;  // the view's light group
  TQ3GroupPosition  thePos;    // a group position
  TQ3Object         theLight;  // a light
  TQ3Status         theResult; // a result code
  TQ3ObjectType     theType;
  TQ3Point3D        theLoc;
  TQ3Vector3D       theDir;


  // Get the light group from the view.
  theResult = Q3View_GetLightGroup(inView, &theGroup);
  if (theResult == kQ3Failure) goto bail;

  // Traverse the light group and mirror the positions
  // and direcitons of all light types as needed.
  for ( Q3Group_GetFirstPosition(theGroup, &thePos);
      thePos != NULL;
      Q3Group_GetNextPosition(theGroup, &thePos))
  {
    theResult = Q3Group_GetPositionObject(theGroup,
                                          thePos,
                                          &theLight);
    if (theResult == kQ3Failure) goto bail;

    theType = Q3Light_GetType(theLight);

    // What we mirror depends on the type of light.
    switch (theType)
    {
      case kQ3LightTypeAmbient: break;

      case kQ3LightTypePoint:

        Q3PointLight_GetLocation(theLight, &theLoc);
        Q3Point3D_Transform(&theLoc, &inMatrix, &theLoc);
        Q3PointLight_SetLocation(theLight, &theLoc);
        break;

      case kQ3LightTypeDirectional:

      Q3DirectionalLight_GetDirection(theLight, &theDir);
        Q3Vector3D_Transform(&theDir,
                             &inMatrix,
                             &theDir);
      Q3DirectionalLight_SetDirection(theLight, &theDir);
        break;
```

```
        case kQ3LightTypeSpot:

            Q3SpotLight_GetLocation(theLight, &theLoc);
            Q3Point3D_Transform(&theLoc, &inMatrix, &theLoc);
            Q3SpotLight_SetLocation(theLight, &theLoc);

            Q3SpotLight_GetDirection(theLight, &theDir);
            Q3Vector3D_Transform( &theDir,
                        &inMatrix,
                        &theDir);
            Q3SpotLight_SetDirection(theLight, &theDir);

            break;
    }

    // balance reference count of light
    Q3Object_Dispose(theLight);
  }

  Q3Object_Dispose(theGroup);

  return(kQ3Success);

  bail:
  return(kQ3Failure);
}
```

Mirroring the three view plane cameras

We now return to Listing 6 AdjustOneCamera to look what happens if we wish to mirror the cameras. Look for the else-branch of the conditional statement if(!gMirrored). First we determine which of the three views we are dealing with. Then we mirror the location of the camera and adjust the CenterXOnViewPlane and CenterYOnViewPlane variables. By now you should be able to relate this code to the mirrored cameras in **Figures 14** to **16**.

Mirroring the virtual scene

Of course we also need to mirror the model which we loaded from disk and the background planes. Look at the conditional statements if(gMirrored) in SubmitViews in Rendering.c. Just before we call SubmitOneView for each view, we set the field fMirrorMatrix of the struct gDoc to the mirroring matrix for that view. In SubmitOneView we submit the matrix in the fMirrorMatrix field before anything else. This is what mirrors the background planes and the model.

Adjusting the orientation style

Think about what we did when we did our mirroring. We mirrored our model, so we mirrored its polygons and its vertices. That means we reversed the direction in which the vertices of a polygon are listed. This is important as the direction in which the vertices of a polygon are listed determines which side of the polygon is considered as the front face which in turn influences shading. We can change what QuickDraw 3D considers to be the front face by changing the orientation style. The default is kQ3OrientationStyleCounterClockwise, which means that the front face is the side from which the vertices are listed counterclockwise. By changing the orientation style to kQ3OrientationStyleClockwise the front and back face are flipped. This is what happens in Listing

8, SubmitOneView in Rendering.c. Notice that we do not explicitly toggle between kQ3OrientationStyleCounterClockwise and kQ3OrientationStyleClockwise. In non-mirrored mode we simply rely on the default. Only in the mirrored mode do we actually submit an orientation style using Q3OrientationStyle_Submit. The style is created in InitDocumentData in (De)Init.c through the call Q3OrientationStyle_New (Listing 12). You may wish to read up on the orientation style object in the QuickDraw 3D 1.5.4 manual (Chapter 6, page 550).

Listing 12 (De)Init.c

InitDocumentData

```
ioDoc->fOrientationStyle =
Q3OrientationStyle_New(kQ3OrientationStyleClockwise);
```

Troubleshooting

Make sure you have a look at the troubleshooting sections of our previous articles in MacTech July 1998 and August 1998. In addition, have a look whether your problem is among these.

One of the background planes remains untextured

Explanation: You may have run out of VRAM. Running in a high resolution (say 1024x768 or above), in millions of colours and a couple of textures may simply be too much for the VRAM of your graphics board, especially if it has only 4MB or 8MB.

Solution: Try running in thousands rather than millions of colours, reduce the sizes of your textures or use fewer textures. If these solutions are out of the question, consider a graphics board with more VRAM.

Tidbits

As always there are ways to improve the code. Here are some things you may wish to look at.

Delays between views

In the MacTech article on single screen head-tracked displays we mentioned the problem caused by delay: the perspective shown on the screen does not correspond to the head position of the user, causing the virtual scene to appear distorted. Cubby's three views cause an additional delay related problem. Because the three views do not render equally fast, we can get discontinuities on the edge of two views. This effect become particularly noticeable under quick camera movements. One way to eliminate this problem would be to use a TQ3PixmapDrawContext. All three views are rendered to an offscreen GWorld and copy this GWorld to the screen. The awkward thing is that not all 3D graphics boards support accelerated rendering to offscreen GWorlds. For graphics accelerators by ATI you can get code from ATI developer support which makes accelerated offscreen rendering possible.

Limiting the camera position

We discussed the problem that we had with the hither plane of a camera being pushed through the background planes, causing the latter to disappear. Just as the delays between the views, this can be fixed by using a TQ3PixmapDrawContext. A 2D background texture is copied to a Gworld which is used as the pixmap. The virtual scene is rendered over this background texture. So instead of making the background planes part of the virtual scene, the background planes are a simple 2D background texture.

Conclusions

By now you should have a good idea of how the visualization part of Cubby works. You learnt how to configure three views with view plane cameras. You also learnt how to mirror the resulting images. But before you can enjoy a virtual scene in Cubby there's some more work to do. You need to know how to build an InputSprocket driver and how to calibrate the head-tracker so that the user's head movements give the correct perspectives in Cubby. Tune in next month and we'll show you how.

Bibliography and References

- Cruz-Neira, C., Sandin, D.J., & DeFanti, T.A. (1993). Surround-screen projection based virtual reality: The design and implementation of the CAVE. *Proceedings of SIGGRAPH'93*, 135-142.
- Fernicola, P. and Thompson, N. (1995, June). *QuickDraw 3D: a New Dimension in Macintosh Graphics.* Develop 22, pp.6-28
- Djajadiningrat, J.P. (1998). Cubby: What you see is where you act. Interlacing the display and manipulation spaces. Doctoral dissertation, Delft University of Technology, Delft.
- Djajadiningrat, J.P., Smets, G.J.F., & Overbeeke, C.J. (1997). Cubby: a multiscreen movement parallax display for direct manual manipulation. *Displays*, 17, 191-197.
- Djajadiningrat, J.P., & Gribnau, M.W. (1998, July). Desktop VR using QuickDraw 3D, Part I. Using the View Plane Camera to implement a head-tracked display. *MacTech*, 14(7), 32-43.
- Greenstone, B. (1995). QuickDraw 3D. In McCornack et al. (eds.), *Tricks of the Mac Game Programming Gurus* (pp. 491-546). Indianapolis, IN: Hayden Books.
- Gribnau, M.W., & Djajadiningrat, J.P. (1998, August). Desktop VR using QuickDraw 3D, Part II. Using the Pointing Device Manager to implement a head-tracked display. *MacTech*, 14(8), 26-34.

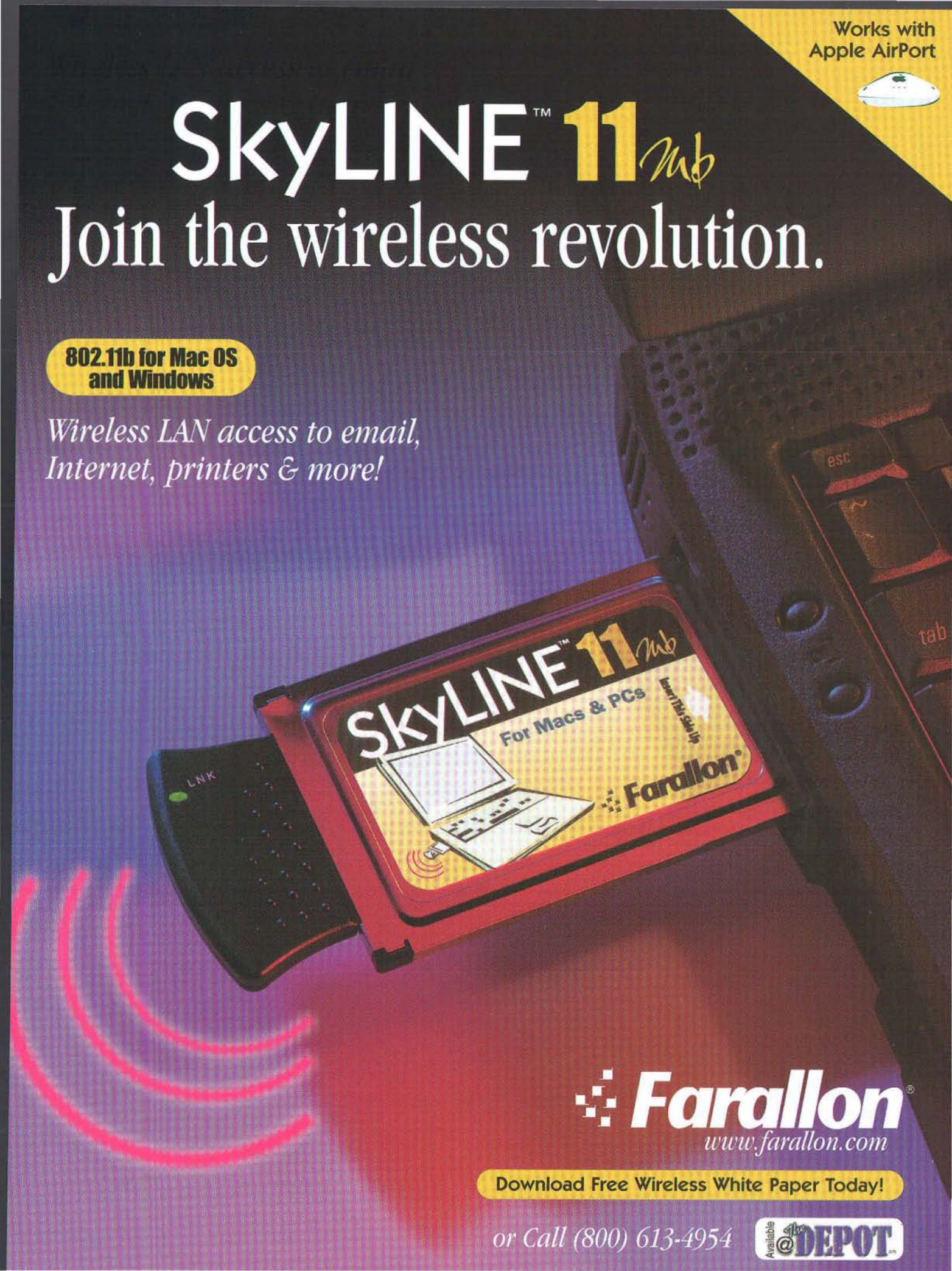
Works with Apple AirPort

SkyLINE™ 11mb

Join the wireless revolution.

802.11b for Mac OS and Windows

Wireless LAN access to email, Internet, printers & more!



Farallon®

www.farallon.com

Download Free Wireless White Paper Today!

or Call (800) 613-4954

Available @the DEPOT

by Jeff Clites <online@mactech.com>

PDF and XML

Last month we covered Adobe's Portable Document Format (PDF), focusing on how it relates to Quartz, Apple's new imaging model. In brief, PDF originated as a simplification of PostScript, retaining PostScript's primitive graphics operators while discarding its programming-language constructs and adding file and document structure specifications. Quartz (specifically, the Core Graphics Rendering API) is again based on this same set of operators, making it natural to "record" graphics operations into a PDF file, and just as natural to "play back" a PDF into a series of native drawing instructions. At its simplest, PDF is the new PICT; more interestingly, the Quartz imaging model is at the center of all 2-D graphics on Mac OS X, providing a centralized facility for rending drawing commands from different APIs (such as QuickDraw) into different output formats, be they destined for the screen, a printer, or a file.

Of course, part of the beauty of Quartz is that it frees the programmer from having to worry about the details of this process. At the same time, Quartz is certain to increase the popularity of PDF, and in particular expand its use beyond just a format for traditional documents. Accordingly, it will be to a programmer's advantage to know as much as possible about PDF, and to be aware of its strengths and its weaknesses.

As touched on above, PDF defines a file format in addition to a graphics model. In the abstract, a PDF file describes a tree of objects, with a significant separation between document content and document layout. This should send off bells in a developer's head, because it sounds similar to XML, and it's natural to wonder how deep this connection is—to ask questions like, "can a PDF document be represented in XML." The short answer is "probably not", but it's interesting to investigate the parallels between the two formats.

Intersections with XML

PDF and XML are similar in that they define a file structure which is designed to encapsulate a wide range of data in a fairly generalized, hierarchical fashion. Although PDF is designed to be extensible, it does define an interpretation for the information it contains, and it's not clear how well current PDF-rendering applications would handle PDF documents with content which they don't recognize. XML is at the other extreme. At its core, it says nothing about the semantics of the data which it can contain, and it's often used as a format for information which isn't naturally thought of as a "document." But given its generality, it would certainly be possible to devise an XML-based format to encapsulate page-descriptions in a manner similar to PDF. On the other hand, there are several facilities of PDF which are not easily mimicked using XML—features dealing more with practical performance issues than with conceptual structure.

PDF was designed to be a *final format*, so that PDFs represent finished documents, rather than in-progress works (such as word-processing documents) which will be extensively changed. Still, it is possible to make limited modifications to PDFs, and interestingly this can be done by appending the "change" information to the end of a PDF, without requiring the entire document to be rewritten. This makes it convenient to prepare an initial document and at a later stage add annotations or hyperlinks. This approach also provides a measure of safety, as previous versions of a document can be recovered simply by truncating the changes off the end, and modifications cannot cause complete corruption of the base document. This also means that it is possible to modify large documents without large resource requirements.

Despite XML's flexibility, it isn't possible to create a well-formed XML document by appending information directly to another document, because of the requirement that there be a single root element. (It is possible to work around this limitation, but only by splitting the document into multiple files.) Additionally, PDF documents frequently encapsulate binary data (such as images or compressed text), and it is not convenient to embed such data into XML documents directly—XML is a text-based format, and binary data could be interpreted as markup, or mangled if the document is converted to a different character encoding. XML-based formats traditionally handle this by storing the data in a separate file which is then referenced from the base document, just as images are included in HTML files. This is less convenient than

PDF's single-file approach. (It would be possible to include binary data in XML documents by converting it into a text-based representation, such as Base-64 encoding, but this tends to offset the benefits of compression.) Finally, PDF has a higher structural flexibility, in that logical containment is not always represented by physical containment. In other words, structures which logically contain other objects may do so by referencing the objects by name, whereas in XML such containment is almost always represented by physically nesting elements. This flexibility allows the same PDF to be represented in different ways, so that for example a PDF file may be optimized for page-at-a-time delivery over the internet, or alternatively it could be created in a single-pass by a printer driver.

FOP

So despite the current popularity of XML, it isn't likely that PDF is going to be superceded any time soon. So where do PDF and XML intersect? Well, as we observed before, it's natural to think of XML as unformatted data, and to think of PDF as an output format. The preferred way to get from XML to something with formatting is by way of XSL Transformations (XSLT). In the case of XML-to-PDF transformation, there's a tool to help with the process, FOP. (It's part of the Apache XML project.) To use FOP, you first use an XSLT processor to convert your XML document into a tree of formatting objects, which may itself be represented as an XML document. This is where you determine the form of your final document. Since, as mentioned above, XML documents are traditionally devoid of formatting information and are often viewed as pure data, any decisions about how this information will be presented must be encapsulated in the style sheet. Once this is done, and you have your tree of formatting objects, you feed this into FOP, which produces your final PDF. FOP is very much a work in progress, and does not yet support all of the formatting objects defined in the XSL specification, but even as-is it appears quite useful. IBM has an informative tutorial on transforming XML documents. (A free registration is required to access the tutorial.) It discusses using FOP to create PDF documents, and in addition shows you how to generate SVG (Scalable Vector Graphics), which is useful for creating things like charts and graphs from XML-encapsulated data.

FOP
<http://xml.apache.org/fop/>
Tutorial: Transforming XML documents
<http://www.ibm.com/software/developer/education/transforming-xml/>

OmniPDF

Finally, while you're playing with PDF, be sure to check out OmniPDF if you are running Mac OS X. It's a very cool PDF viewer. It's still under development, but it's Cocoa-native (and hence Mac-OS-X-native), and it really shows off the power of Quartz, as it uses Core Graphics Rendering to do its magic. (OmniPDF is from the Omni Group, who also created OmniWeb, which is currently the only Cocoa-native web browser available. You should check it out also—it's a refreshing alternative, and it has many fun features which set it apart from your usual browser choices.)

OmniPDF
<http://www.omnigroup.com/products/omnipdf/>
OmniWeb
<http://www.omnigroup.com/products/omniweb/> MT

By Larry Taylor Edited by Steve & Patricia Sheets

Getting Started with Perl

Open Source power scripting for Macs.

Introduction

Perl is a programming/scripting language developed under Unix, which is distributed under the GNU license and now runs on most platforms, including MacOS. It is the language of choice for Unix system administration, CGI scripts and other goodies. More relevantly, it can really expand your ability to accomplish things on the Mac. In this article I describe a frustrating problem I had and a step by step Perl solution. I hope this example will encourage you to learn Perl and use it. Perl scripts are just text files and so are fairly easily portable across platforms making Perl even more useful if you need to solve the same problem on several platforms. Learning Perl is not difficult and it looks great on your resume, so why not give it a try?

Mac + Perl = MacPerl

Perl arouse because many UNIX programmers wanted a quick alternative to C, with many of C's features. The result was a full-featured, easy to use, C-like programming language. Perl has been ported to the Mac where it can be used to create pseudo-applications called droplets. I call them pseudo because they do not have individual types and creators and so they must either be opened by double clicking or by dragging a document onto them. They are interpreted and so need the Perl interpreter in order to run. No Mac interface is needed to get information in or out, so Perl is ideal for projects that involve reading some data, analyzing it, and outputting some conclusions, projects for which the event-loop paradigm is more of an annoyance then a help (although Cmd-period will stop runaway Perl droplets). One can construct compiled applications with a full Mac interface, but the files are large and the advantages over C largely evaporate. I use Perl for tasks as varied as extracting data from files to emailing students in a class their exam scores.

Perl is "open source" software. The interpreter is available to download for free at <http://www.iis.ee.ethz.ch/~neeri/macintosh/perl.html>, or the book "MacPerl, Power and Ease" by Vicki Brown and Chris Nandor (#1-881957-32-2) from Prime Time Freeware <www.ptf.com> contains a CD with the interpreter and lots of other useful stuff. The book itself is a nice introduction to programming in general and Perl in particular. Additional Perl stuff can be gleaned from the net. Try starting at <http://www.perl.com>.

The Problem

Got one of those cool digital cameras that saves images to floppies? Then you know the files are labeled automatically, MVC-01L.JPG, MVC-02L.JPG, etc. Copy the images to your computer and you're in business. But suppose you went wild and filled up two disks? Or ten? Files on different floppies often have the same name, so you can't just copy them to the same folder. So you copy one floppy, change all the names of the files, copy the second, etc. – bummer. Even with just a few images, you tend to put them in a folder with a useful name since otherwise you won't remember what the pictures are about, can't search for them with Sherlock, etc. Wouldn't it be nice to have them named, whatever1.jpg, whatever2.jpg, etc.? This is a perfect job for a script.

The script should begin with a folder named whatever and look inside it for all the MVC files and rename them as whatever1.jpg, whatever2.jpg, etc. It should even be a bit smarter.

Larry Taylor is a research mathematician and professor who spends too much time fooling around with this sort of thing. More stuff at www.nd.edu/~taylor.

If there are going to be ten or more, the first should be whatever01.jpg: if there are 100 or more, whatever001.jpg, if there are ... but you get the idea. Even more, if there are already some whatever files, it should number the new MVC files to fit into the pattern. Specifically, it should look at the creation time of the first MVC file and the first whatever file. If the MVC time is later, the MVC files should come after the whatever files, but otherwise the whatever files should be renamed and the MVC files should come first. If the user trashed a few of the whatever files so they are no longer in sequence, the whatever files should be renamed so as to be in sequence.

Using this script, you can copy one disk worth of images into a folder, run the script, copy the next disk, run the script, etc. At any time during the process, the images can be viewed and those that are unwanted can be deleted.. At the end, all of the "keepers" are named consecutively in the order in which they were taken, no matter the order in which they are copied or removed.

The Script

Open the MacPerl application and select New from the File menu and you're ready to start. Line 1 should be #!perl. This is a holdover from the Unix world where this line tells the operating system to feed this file to the Perl interpreter. You can also do things with it in MacPerl, but we don't here. Now save the file. Name it what you will. At the bottom of the dialog box is a pop up menu labeled "Type:" (reading "Plain Text"). Set the menu to "Droplet" and save.

The advantage of a droplet is that you can just drop items onto its icon and the information is passed on to the script. In this script we include no other way to input folder/file information, although Perl can do so, even through standard file. The folder/file information is passed to the script as $ARGV[0]for the first folder/file, $ARGV[1] for the second, etc. Droplets allow us to use the Mac GUI to mimic the command line paradigm. Dropping a collection of files on a droplet has the same effect as the command line, droplet_name file1 file2 ...

Before discussing the code, here is an outline for solving the problem.

Step 1: Get the folder name. If a folder is dropped, use it; if a file is dropped, use the enclosing folder. If several items are dropped, process them all.

Step 2: Collect the names of the MVC files and the whatever files.

Step 3: Get the two creation times and figure out the starting numbers for the two sets of files.

Step 4: Rename the files.

We do a certain amount of error checking and quit at the first sign of trouble – these may be your only photos of Aunt Rose. Perl borrows much from C, including the tendency to write short functions (subroutines in Perl). One immediate difference is the lack of variable typing (the same variable can be a number or a string, depending on context). Another is the ability to work with arrays whose size is unknown before execution, As a language, Perl is

particularly adept at manipulating arrays and strings and it does file management rather well.

Now for the code. We write a sequence of subroutines most of which just do one of the steps outlined above and pass the relevant data on to the next. We try to introduce some interesting features of Perl in discussing each subroutine. More information can be gleaned from the code and its comments. Here is the first routine. The for loop works its way through the dropped items, passing each one in turn to the subroutine do_a_folder which returns false if anything goes wrong. Ordinary Perl variable names start with $; arrays start with @; $#foo is the last index of the array @foo. As with C, the first array element is $foo[0]. If this were C the braces would be optional, but in Perl they are required.

```
for($ii=0;$ii<=$#ARGV; $ii++) { # This is a Perl comment.
  if(!do_a_folder($ARGV[$ii])){exit;}
  }
```

Perl handles file system objects via path names and the $ARGV variables are path names. The first line of the subroutine illustrates the way Perl passes variables to subroutines: the values are in a list/stack named @_ and we can shift them off in order. The rest of the routine is straightforward. Perl has a simple syntax for checking if strings are folders or files, using two simple "if" tests. One wrinkle here is that if you drop two MVC files on the droplet, by the time the second one is ready to be processed, it no longer exists since it was renamed on the first pass. The routine does nothing in this case except return true, which is what we want. In short, this subroutine handles Step 1 for each dropped object and passes the results to the next subroutine.

```
sub do_a_folder{
$object=shift(@_);
if( -f $object) { # -f checks if $object is a file,
                  # if it is, get enclosing folder.
  $x=rindex($object,':');  # find LAST occurrence of :
  $object=substr($object,0,$x); # remove last part of
                                # path name
  }
# $object now path name to folder
$x=rindex($object,':');  # find LAST occurrence of :
$fold_name=substr($object,$x+1);# get name of folder
if( -d $object) { # it's a folder
  unlink("$object:MAVICA.HTM");
    # This deletes a junk file which often gets copied.
  return process_folder($object,$fold_name);
  }
  # else quietly do nothing.
return 1;
}
```

Extract the relevant files into two arrays. There is no need to specify the size of these arrays in advance since Perl handles these details. The undef's make sure that these arrays are empty at the start. Explicitly initializing variables is usually a good idea. One outstanding feature of Perl is Unix regular expression matching and substitution. Look how easy it is to find the files we want:

```
if( $files[$i]=~m/^$fold_name\d*\.jpg$/)
```

This is true if the string on the left contains the expression between the /'s. That expression says the string must begin (^) with $fold_name, have any number of digits (\d*) and then end ($) with a .jpg. The dot is \. because . means match any character. When we find a file of the desired type, the push puts it at the end of the appropriate array. Note that the elseif of C becomes elsif. Finally, the construction \@mvc_files is a way to pass a reference to the entire array to the next subroutine.

```
sub process_folder{
$fold=shift(@_);
$fold_name=shift(@_);
# Make sure names can't be too long for the Finder.
$fold_name=substr($fold_name,0,23);
undef(@fold_name_files); # Clear old values
undef(@fold_name_files); # Clear old values
if( opendir(DIR,$fold)) {# if we can read the directory
  chdir($fold); # change the working directory
  @files=readdir(DIR); # read all objects into an array
  closedir(DIR);  # close the directory for reading
  for($i=0;$i<=$#files;$i++) {
    if( $files[$i]=~m/^$fold_name\d*\.jpg$/){
        # remember the folder_name files
      push(@fold_name_files,$files[$i]);
      }
    elsif( $files[$i]=~m/^MVC-\d*L\.JPG$/) {
        # remember the MVC files
      push(@mvc_files,$files[$i]);
      }
    }
  if($#mvc_files<0 && $#fold_name_files<0) {
    return 1; # Nothing to do.
    }
  else { # Go rename the files.
  return (
    setup_rename(\@mvc_files,\@fold_name_files,$fold_name));
    }
  }
else { print"Failed to open $fold\n"; return 0;}
}
```

In the first few lines of the next subroutine, we retrieve the reference to the arrays. The syntax is straightforward: in the previous subroutine @mvc_files was an array: in this subroutine the same array is @$mvc_files. There is no need to use the same name.

Now look at the phrase:

```
length($#$fold_name_files+$#$mvc_files+1+$startNumber)
```

This is an example of how variable type changes: $#$fold_name_files is one less than the number of files in the array @$fold_name_files so the sum is the biggest number in a file name. The function length treats the number as a string and returns its length. If we have more than 9,999 files, we quit since then the file names might be longer than the Finder limit of 31 characters.

Perl has built-in functions to easily extract file information. We have no trouble getting creation times: the function stat returns an array of data and the eleventh element in the array is the creation time. Remember, the first is [0]. We then use this information to determine the starting number for the two sets of file names. This completes Step 3 and we pass the needed information on to the next subroutine.

```
sub setup_rename{
$mvc_files=shift(@_);
$fold_name_files=shift(@_);
$fold_name=shift(@_);
$startNumber=1; # The first file is numbered 1.
#
$new_digit_size=length(
      $#$fold_name_files+$#$mvc_files+1+$startNumber);
if($new_digit_size>4){
  print"More than 9,999 files? No way!\n";
  return 1;# Will process other folders
  }
#
```

```
# Get MVC creation time (if possible).
if( ($#$fold_name_files>=0) ) {
  $time_MVC=(stat($$mvc_files[0]))[10];
  }
# Get folder_name creation time (if possible).
if($#$fold_name_files>=0) {
  $time_FN=(stat($$fold_name_files[0]))[10];
  }
# Calculate starting numbers.
if($#$mvc_files<0) { $fold_name_startNumber=$startNumber;}
elsif($#$fold_name_files<0) {$mvc_startNumber=$startNumber;}
elsif($time_MVC<$time_FN) {
  $mvc_startNumber=$startNumber;
  $fold_name_startNumber=$#$mvc_files+1+$startNumber;
  }
else {
  $mvc_startNumber=$#$fold_name_files+1+$startNumber;
  $fold_name_startNumber=$startNumber;
  }
return rename_files($mvc_files,$mvc_startNumber,
    $fold_name_files,$fold_name_startNumber,
    $fold_name,$new_digit_size);
}
```

The rename routine (Step 4) is a bit more complicated. The Perl rename routine is a Unix style routine, so if there already is a file with the new name, the old file is destroyed without warning. The Mac solution is better, but annoying – put up a dialog box and let the user recover. But you don't want dialog boxes, you just want the files renamed. The solution we use is to create a temporary folder, move the files into this folder as we rename them, move them back when we are done, and finally, delete the temporary folder. We put this temporary folder in our enclosing folder so that in the event of an error it should be easy to find all your files.

Here we introduce another way to collect the information passed as the arguments: make a list on the left and set it equal to @_. The mkdir, rmdir functions betray their Unix heritage. Subroutines move the files into the temporary folder and out of it again.

```
sub rename_files{
# Make temporary folder - the name will be a number
$dir=0;
while( -d $dir || -f $dir ) {$dir++;}
  # Possible infinite loop - but need thousands of
  # folders/files with numbers as names.Don't worry.
if(!mkdir($dir,0777)) {
  print"Failed to make temporary folder.\n";
  return 0;
  }
($filesA,$startA,$filesB,$startB,$prefix,$digit_size)=@_;
$dir_prefix=":$dir:$prefix";
# Move the first batch of files, then the second.
# Bail if error.
if(!mv_tmp($startA,$filesA,$dir_prefix,$digit_size)){
  return 0;
  }
if(!mv_tmp($startB,$filesB,$dir_prefix,$digit_size)){
  return 0;
  }
# move the files back. Bail if error.
if(!mv_back($dir)){return 0;}
# Delete the temporary directory
return rmdir($dir);
}
```

Nothing much new in the next subroutine except the foreach loop. This works through the array setting $h to the values of the array in order – no need for an index variable. This is not earthshaking, but elegant. The s routine completes the script.

```
sub mv_tmp{
($first,$list,$dir_prefix,$digitSize)=@_;
foreach $h (@$list) {
  $numStr=substr("00000",0,$digitSize-length($first)).$first;
  if(!rename($h,"$dir_prefix$numStr.jpg") ){
    print"Failed to move $h into $dir\n";
    return 0;
    }
    $first++;
  }
return 1;
}

sub mv_back{
$dir=shift(@_);
if(opendir(DIR,$dir) ){
  @files=readdir(DIR); # read all objects into an array
  closedir(DIR);  # close the directory for reading
  chdir($dir);
  foreach $h (@files) {
    if(!rename($h,"::$h") ){
      print"Failed to move $h out of $dir\n";
      return 0;
      }
    }
  chdir("::");
  return 1;
  }
else {return 0;}
}
```

Final Comments

The constructions, syntax and built-in functions discussed in this short article have barely scratched the surface of what is available. And more is coming every day. See <http://www.perl.com> and related links. I hope this example will spark your interest in using Perl for your own projects. Happy scripting.

ADVERTISER/ PRODUCT LIST

List of Advertisers

List of Products

Mac OS X Porting and Development Showcase

Mac OS X Porting and Development Showcase

HOW DO YOu SAY SuPERB IN LINuX?

SuSE.

{soo´ sah} is {soo´ pûrb}

When you think of Superb, you think of unusually high quality. Like a superb wine, for example. Majestic. Rich. Luxurious. Superior. ■ SuSE Linux is all that. And more. More experience. More adaptability. More applications—over 1500. ■ More power to you. And more freedom, too. ■ No wonder more than 50,000 enterprises worldwide bank on its superb open source solutions. Making SuSE the international Linux leader—setting a higher standard for excellence, simplicity and support. ■ Even the price is superb. ■ So, how do you say superb in Linux? SuSE. It's a lesson well learned.

www.SuSE.com

Versions for Intel, Alpha, and PowerPC

The freedom to change.
The power to change the Linux world.

SuSE

Time matters.

The fate of the company is in your hands.

Every second counts when you're competing at Internet speed. And the faster your Web application is developed, deployed and maintained, the faster your competition will drop out of sight.

The solution? Use Pervasive Software's Tango 2000 to develop your ideas with double the speed of any other development software.

- Visually develop applications on Mac, and deploy on Mac, Windows, Linux and Solaris
- Advanced XML support
- Connect directly to Filemaker or any ODBC database
- Extend applications with JavaBeans
- High performance Tango server scales with cluster support

Give yourself the same competitive edge that Tango has given leading companies like Apple Computer®, theglobe.com™ and Harbor Freight Tools™.

Download your Free Tango Demo today!

Grab the time-saving advantages of Tango 2000 absolutely FREE. Visit our Web site and download our FREE fully functional Tango 2000 demo. Or call us now to get your demo on CD.

www.pervasive.com/speed or **1-800-287-4383**

Hurry! This offer ends soon.

PERVASIVE SOFTWARE

The Freedom to Create Applications™ for Everyone, Everywhere